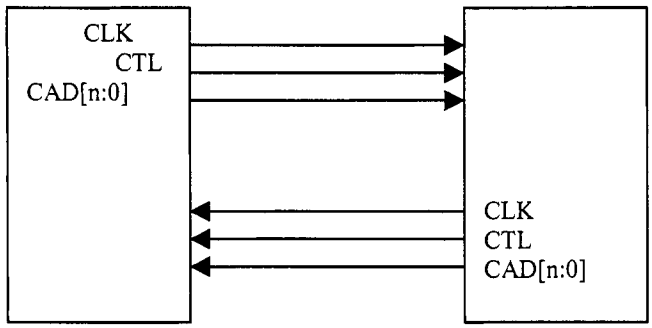
Lightning Data Transport IO Specification

Revision 0.17

AMD Company Confidential

1 Overview

This document describes the Lightning Data Transport (LDT) I/O link. LDT is a packet-based link implemented on 2 unidirectional sets of wires. The link is packet-based, nominally point-to-point, and connects exactly two devices. Devices may have multiple LDT links, allowing the construction of larger LDT fabrics.



LDT is intended to be used as an I/O channel, connecting chains of LDT I/O devices and bridges to a host system. The interface from the host to the LDT chain(s) is called the host bridge.

1.1 LDT Terminology

- A ***cycle*** is a period of time defined by the period of the clock (rising edge of CLK to rising edge of CLK)
- A ***bit time*** is half a clock period in duration – two data bits are transmitted on each wire per cycle.
- A ***node*** is a physical entity that connects to one end of an LDT link.
- A ***packet*** is a series of bit times and forms the basis of communication between two nodes.
- A ***transaction*** is a sequence of packets exchanged between two or more nodes in the system which results in a transfer of information.
- A ***source*** is the node that starts a transaction.
- A ***target*** is the node that ultimately services the transaction on behalf of the source. Note that there may be intermediary nodes between the source and the destination.
- A ***unit*** or ***function*** is a logical entity within a node that may act as a source or destination of transactions. Functions are useful for describing the transaction ordering rules for LDT.
- A ***doubleword (DW)*** is four bytes.
- A ***quadword (QW)*** is eight bytes.

2 LDT Signals

Signal	Width	Description
CAD	8, 16, or 32	**C**ommand / **A**ddress / **D**ata. Carries LDT commands, responses, addresses and data. CAD width may be different in each direction.
CTL	1	When asserted, CAD carry a control packet. When deasserted CAD carry data.
CLK	1, 2, or 4	Clocks for the above signals. Each byte of CAD has a separate clock signal. Regardless of the width of the link, CTL is clocked by the same CLK as CAD[7:0].

The signals given in the table constitute a single unidirectional connection between two nodes. A full link requires a connection in each direction; however, the connections need not be the same width in each direction.

LDT links wider than 8 bits are built by ganging multiple 8 bit links in parallel to form either 16 or 32 bit links. Links wider than 8 bits have one clock per byte, but still only one CTL bit for the whole link. The forwarded clock for the CTL signal is the clock for the least significant byte.

In addition to the link signals, all LDT devices require the following reset/initialization input pins:

Signal	Width	Description
PwrOK	1	Power and clocks are stable.
Reset_L	1	Reset the LDT chain

All devices in a given LDT chain will receive the same PwrOK and Reset_L signals. These signals are open drain wired-OR signals, thus allowing multiple sources (possibly including host bridges) to initiate a reset sequence. See section 13 for information on reset sequencing.

3 Packet Definition

This section describes the packet definition for the LDT link. All figures shown in this section assume an 8 bit wide link.

The packet structure for 16 and 32 bit links can be derived from the 8-bit link packet structure by combining the fields within adjacent bit times. Some examples:

$BT1_{16}[15:0] = BT2_8[7:0]$ & $BT1_8[7:0]$

$BT1_{32}[31:0] = BT4_8[7:0]$ & $BT3_8[7:0]$ & $BT2_8[7:0]$ & $BT1_8[7:0]$

Where BTN_m represents the Nth bit time within a packet for an LDT link of width m and "&" represents concatenation.

Since all packets are multiples of 4 bytes long, packet boundaries will always fall on bit-time boundaries in the 16 and 32 bit case, as well as the 8 bit case.

3.1 Use of the CTL Signal

LDT consists of control packets and data packets, distinguished by the use of the CTL signal. Link transmitters assert CTL during all bit times of control packets, and deassert it during data packets. The purpose of the CTL signal is to allow control packets to be inserted in the middle of long data packets, and to stall in the middle of control packets.

Here are the rules that govern packet transmission.

1. CTL is asserted through all bit times of a control packet.
2. CTL may be deasserted on 4-byte boundaries within a control packet, to insert stalls. No valid information is transmitted on the link during the stall. When CTL is reasserted, the interrupted control packet continues from where it left off.
3. CTL is deasserted through all bit times of a data packet.
4. CTL may be asserted on 4-byte boundaries within a data packet to insert a control packet. Control packets inserted into data packets must not themselves have an associated data packet. CTL may still be deasserted on 4-byte boundaries within the control packet to cause stalls. When CTL is deasserted at the conclusion of a control packet, data transfer continues from the point where it left off.
5. Write command and read response packets always have an associated data packet. The data packet may not immediately follow the last bit time of its associated control packet, as other control packets may be inserted before the data packet. However, since these control packets may not have associated data, there is only ever one data transfer outstanding.
6. The order of operations on the link is determined by the order of the control packets. The fact that data transfer for a control packet may be delayed does not affect how it is ordered.

7. The bit time immediately following the last bit-time of a data packet is always the start of a control packet (CTL must be asserted).
8. CTL may only be asserted or deasserted on a 4 byte boundary.
9. CTL may only be deasserted when data transfer due to a previously transmitted control packet is pending, or in the middle of a control packet.

3.2 Packet Structure

This section defines the basic control and data packet types, and shows the position of the fields that are common to all the control packet types. All packets are multiples of 4 bytes long.

3.2.1 Control Packets

Control packets consists of 4 or 8 bytes. This section shows the basic structure of each of these control packet forms.

In the diagrams that follow, the unlabelled packet fields are command-specific. Some common control packet fields are:

- ***Cmd[5:0]*** is the command field which defines the packet type.
- ***UnitID[4:0]*** serves to identify participants in a transaction. Since all packets are transferred either to or from the host bridge at the end of the fabric, either the source or destination node is implied. The value zero is reserved for the UnitID of the host bridge. For requests, UnitID denotes the source of the request. For responses travelling downstream, UnitID denotes the node to which the response is being sent. For responses travelling upstream, UnitID denotes the node that generated the response. Nodes with multiple logical I/O streams may own multiple UnitID values.
- ***Bridge*** indicates that this response packet was placed onto the link by the host bridge, and is used to distinguish responses travelling upstream from responses travelling downstream. In the case of two host bridges sending packets to each other on a double-ended chain, the target host bridge appears to the requesting host bridge as an LDT slave device. Therefore, the bridge bit will be clear on responses to requests issued from the far host bridge.
- ***SeqId[3:0]*** is used to tag groups of requests which were issued as part of an ordered sequence by a device, and must be strongly ordered within a virtual channel. All requests within the same I/O stream and virtual channel that have matching nonzero seqId fields must have their ordering maintained. The seqId value of 0x0 is reserved to mean that a transaction is not part of a sequence. Transactions with this value have no sequence ordering restrictions, although they may be ordered for other reasons.
- ***PassPW*** indicates that that this packet is allowed to pass packets in the posted request channel in the same I/O stream. Otherwise, it must stay ordered behind them.
- ***SrcTag[4:0]*** is a transaction tag which is used to uniquely identify all transactions in progress from a single requester. Each unitID can have up to 32 transactions in progress at one time. The concatenation of source UnitID and SrcTag serves to uniquely identify nonposted requests.
- ***Addr[39:2]*** represents the doubleword address accessed by the command. Not all address bits are included in all command types. Where finer granularity is required, byte masks are used.

Reserved fields in command packets must always be driven to 0 by transmitters, and must be assumed to be undefined by receivers.

3.2.1.1 Info Packet

Info packets are always 4 bytes long. They are used for nearest neighbor communication between nodes, and so exist at a lower level of the protocol than anything else. They are not routed within the fabric, and require no buffering in the nodes. They are not flow-controlled, and can always be accepted by their destination.

Bit Time	7	6	5	4	3	2	1	0
0	*Command-Specific*		Cmd[5:0]					
1	*Command-Specific*							

2	*Command-Specific*
3	*Command-Specific*

3.2.1.2 Command Packet

Command packets are either 4 or 8 bytes long, depending upon whether the command has an associated address. The diagram below shows a command packet with an address. 4-byte command packets do not contain the address field.

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]	UnitID[4:0]					
2	*Command-Specific*							
3	*Command-Specific*							
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

3.2.1.3 Response Packet

Response packets are always 4 bytes long.

Bit Time	7	6	5	4	3	2	1	0
0	*Command-Specific*		Cmd[5:0]					
1	PassPW	Bridge	*Cmd-Specific*	UnitID[4:0]				
2	Rsv		Error	*Command-Specific*				
3	Rsv		NXA	Rsv				*Command-Specific*

The Error bit is present in all responses, and used to indicate that the issued request could not be completed. If the error bit is set, the NXA (Non-eXistent Address) bit is valid. If it is set, that means that the request could not be completed because no agent on the chain accepted the request. If the bit is clear, it means that the request packet reached its addressed target, but could not be completed by the device.

3.2.1.4 Command Field Encodings

The command field is valid for all control packets.

Code	VChan	Command	Comments/Options	Packet Type
000000	-	Nop	Null packet. Contains flow control info	Info
000001		Reserved		
000010	NPC	Flush	Flush posted writes	Command
000011 0001xx		Reserved		

Code	VChan	Command	Comments/Options	Packet Type
x01xxx x01xxx	NPC or PC (bit 5)	Wr (sized)	Write Command [5] defines whether request is posted. 0: Non-Posted 1: Posted [2] defines the data length. 0: Byte 1: Doubleword [1] defines bandwidth/latency requirements. 0: Normal 1: Isochronous [0] indicates whether access requires host cache coherence (reserved if access is not to host memory). 0: Noncoherent 1: Coherent	Cmd/Addr/Data
01xxxx 01xxxx	NPC	Rd (sized)	Read Commands [3] define ordering requirements for response. 0: Response may not pass posted requests 1: Response may pass posted requests [2] defines the data length. 0: Byte 1: Doubleword [1] defines bandwidth/latency requirements. 0: Normal 1: Isochronous [0] indicates whether access requires host cache coherence (reserved if access is not to host memory). 0: Noncoherent 1: Coherent	Cmd/Address
100xxx		Reserved		
110000	R	RdResponse	Read Response.	Resp/Data
110001 110010		Reserved		
110011	R	TgtDone	Tell source of command that target is done.	Response
1101xx 11100x		Reserved		
11101x	PC or NPC (bit 0)	Broadcast	Broadcast Message. [0] indicates whether a response is required 0: No response required 1: Response required	Cmd/Address
111100	PC	Fence	Fence for posted commands.	Command
111101 111110		Reserved		
111111	-	Sync	Link Synchronization Packet	Info

The fields in the table are as follows:

- ***Code*** is the 6 bit command encoding in each packet.
- ***VChan*** indicates the virtual channel that the packet travels in. Info packets are only used for single-link communication and don't use buffer space, and thus are not in a virtual channel. See sections 4.6 and 4.6 for more information. **PC** = Posted Command, **NPC** = Non-Posted Command, **R** = Response.
- ***Command*** is the mnemonic used to represent the command.

- ***Comments/Options*** gives a short description of the command, and enumerates any option bits within the ***Code*** field.
- ***Packet Type*** indicates the type of packet(s) that this command uses.

3.2.2 Data Packet

Data packets contain the data payload for transactions. Data packets follow write command and read response packets. Data packets range in length from 4 to 64 bytes, in multiples of 4 bytes (1 doubleword). Within a doubleword, data bytes appear in their natural byte lanes. For transfers of less than a full doubleword, the data will be padded with undefined bytes to achieve this byte lane alignment.

This example shows an 8-byte data packet

Bit Time	7	6	5	4	3	2	1	0
0	Data[7:0]							
1	Data[15:8]							
2	Data[23:16]							
3	Data[31:24]							
4	Data[39:32]							
5	Data[47:40]							
6	Data[55:48]							
7	Data[63:56]							

The data packet for a sized read response is arranged with the lowest addressed doubleword returned first and the remainder of the addressed data returned in ascending address order by doubleword. Sized read responses may contain any number of contiguous doublewords within a 64 byte aligned block, although not all bytes are guaranteed to be valid.

Sized doubleword writes work in the same way as sized read responses, and may contain anywhere from 1 to 16 DW, in ascending address order.

Sized byte writes transmit one DW worth of masks first, followed by from 0 to 8 DW of data in ascending address order. Mask[0] corresponds to Data[7:0], Mask[1] to Data[15:8], and so on. 32 mask bits are always transmitted, regardless of the amount of data. All zero byte masks are permitted; however, if any byte masks are nonzero, there must be at least one set mask bit associated with the first doubleword of data.

Bit Time	7	6	5	4	3	2	1	0
0	Mask[7:0]							
1	Mask[15:8]							
2	Mask[23:16]							
3	Mask[31:24]							
4	Data[7:0]							
5	Data[15:8]							
6	Data[23:16]							
7	Data[31:24]							
8+	Packet may contain up to 8 DWs of data							

4 Fabric Operation

The LDT link is a pipelined split transaction interconnect in which transactions are tagged by the source and responses can return to the source out of order. This section outlines the basic operation of the link.

4.1 Topology

LDT I/O fabrics are implemented as a daisy chain of LDT devices, with a bridge to the host system at one end. Devices may implement either one or two links; single-link devices must always sit on the end of the chain, so only one single-link device is possible in a fabric. Direct peer-to-peer communication between devices in the chain is not allowed; all packets (except for info packets) travel between one device and the host bridge. This means that at a high level, the fabric appears as a group of devices directly connected to the host bridge, but not to each other. Packets flowing into the fabric from a host bridge are said to be flowing "downstream". Packets flowing to a host bridge from an LDT device are said to be flowing "upstream".

For convenience in integrating multiple functions onto a single chip, or to allow parallelism between independent request streams, individual LDT devices may use multiple UnitID values. There is no specific limit on the number of physical devices; however, there are only 31 available UnitIDs. No combination of devices that exceeds 31 UnitIDs may be connected to a single link.

Physically, the link may be connected to a host bridge at each end, as long as the fabric contains no single-link devices. This may be useful to provide another path to I/O devices in the event of a host bridge or link failure, or to allow sharing of I/O devices between independent hosts to implement clustering. One bridge will be designated the "master" bridge for the shared link, while the other will be the "slave" bridge. This designation must be made by hardware before the link is reset; the method of doing so is beyond the scope of this specification.

Logically, a double-ended fabric will appear as two distinct daisy-chains, each attached to only one host bridge. By default, the reset sequence will cause all devices to belong to the master host bridge. In the event of a node or link failure, the reset sequence will cause the devices on each side of the break in the fabric to belong to the host bridge on that end. Software may also reconfigure the devices to divide them more evenly between the bridges, to balance traffic.

All devices on the link will be programmed to think of the direction of their owning host bridge as "upstream". Devices are only required to accept requests from the upstream direction. Unaccepted requests continue down the fabric and get an NXA error response when they reach the end. If the device accepts requests from both directions, it must keep track of which link incoming request packets were received on, and send any response back on the same link. An interior node may see the same SrcTag active from the host bridges at both ends of the link. The node must recognize the two host bridges as having disjoint SrcTag spaces. Intelligent devices may also potentially be built that can override their default upstream direction and send requests in both directions. The reasons that a device would do this and its method for determining which direction to send requests are beyond the scope of this specification. If a device sends packets in different directions in the fabric, the fabric cannot make any guarantees about ordering between them.

4.2 Synchronization Packets

Bit Time	7	6	5	4	3	2	1	0
0	11		Cmd[5:0]					
1	11111111							
2	11111111							
3	11111111							

Sync packets are used to indicate a resynchronization event has occurred in the system, such as a reset or a fabric error, which requires all links to be resynchronized. Even though it is defined as a packet type, sync really happens independently on each byte lane in the link.

CRC checking on a link is shut down when a Sync packet is received.

All fields in a Sync packet (including the command) are all 1's. This guarantees that when multiple sync packets are transmitted, the destination will recognize a sync packet even if the bit time counters on both sides have gotten out of step. Therefore, Sync packets will always be transmitted in bursts long enough to guarantee 8 bit times of all 1's in all byte lanes, independent of the width of the link.

BOZO: It seems like we have an issue with how we recognize Sync packets. In our (and it seems like all) implementations, Sync is not actually recognized by the regular command decode hardware, but by logic right in the LDT receiver before the synchronization fifo. This logic must look for more than 4 bit times of all 1's; otherwise, an address packet of all 1's could trigger a sync sequence. Furthermore, in a multi-byte lane link, each receiver is seeing only every other or every fourth bit time, so it would seem like there are lots of legal sequences which could appear as sync packets. How exactly we recognize a sync sequence needs to be determined and spec'ed here.

4.3 Commands

4.3.1 Sized Reads and Writes

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Mask/Count[1:0]		Compat	SrcTag[4:0]/Rsv				
3	Addr[7:2]						Mask/Count[3:2]	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

Sources use the sized read and write commands (Byte, DW) to make requests to either memory or I/O. The data returned for sized reads cannot be coherently cached, as LDT I/O provides no coherence primitives. Sized commands contain the starting doubleword address of the data and a set of data elements to be transferred. Bit 2 of the command indicates whether the data elements to be transferred are bytes or doublewords.

Doubleword operations can transfer any number of contiguous complete doublewords within a 64 byte aligned block. The Count field encodes the number of doubleword data elements that should be transferred, beginning at the specified address, and going in ascending order. Count codes of 0 through 15 represent 1 through 16 data elements to be transferred, respectively. Requests that cross a 64 byte boundary must be broken into multiple transactions on LDT, issued in ascending address order.

Byte reads can transfer any combination of bytes within an aligned doubleword. The Mask field is used to indicate which bytes within the doubleword are being read. If it is desired to issue byte-maskable reads that cross an aligned doubleword boundary, they must be broken into multiple requests, each within a single doubleword. The mask bits may be ignored for reads to regions where reads are guaranteed not to have side effects. A read for which all mask bits are zero can still cause host coherence action (if to memory, and cmd[0] asserted) and still returns a RdResponse packet with 1 DW of (invalid) data.

Byte writes can transfer any combination of bytes within a naturally aligned 32-byte address region. Transfers that cross an aligned 32-byte boundary must be broken into multiple LDT transactions, issued in ascending address order. Address bits [4:2] identify the first double word of data sent in the data packet. The data packet for a byte write operation contains byte mask information in the first DW of the data packet. The Count field is used to indicate the total size of the data packet in DW, including the byte masks, so it will range from 0 to 8 to indicate 1 through 9 DW to be transferred (note that it is possible for

byte writes to contain only the byte masks and so transfer only a single DW). See section 3.2.2 for the format of the data packet. The count field specifies the length of the data packet independent of the value of the byte masks. Nonzero byte masks for double words which are not sent result in undefined behavior. Byte masks may be zero for double words which are sent. The entire byte mask doubleword may be zero, in which case the system performs all activities usually associated with the command; however, no data is written. If any byte masks are nonzero, the byte masks associated with the first doubleword of data must be nonzero.

The sized command field contains a bit that indicates whether or not the access requires coherence action to be taken by the system for host memory accesses. If set, the host must take whatever action is appropriate to guarantee that any caching agent remains coherent with system memory. Writes must cause caches to be updated or invalidated; reads must return the latest modified copy of the data, even if main memory is stale. If the bit is clear, reads and writes may happen directly to/from main memory, without polling or modifying cache states.

Transactions also have an "Isochronous" bit associated with them. See section 6 for details of how this is used.

BOZO: If we never wind up doing anything with isochronous traffic, delete the above statement.
Sized writes have a "posted" bit. Besides serving as a virtual channel identifier, a set "posted" bit indicates that the write request will receive no response in the fabric. The requester's buffer may be deallocated as soon as the write is transmitted. As such, the SrcTag field is reserved for posted requests. No assumptions may be made about the uniqueness of SrcTags for posted requests, either relative to other posted requests or other traffic.

Reads have a "response may pass posted writes" bit. This is carried with the request, but doesn't serve any purpose until the response is generated. At that time, it becomes the PassPW bit in the response. Since writes do not carry the "response may pass posted writes" bit, write responses are always ordered behind posted writes.

The "Compat" bit is used to implement the subtractive decode necessary for legacy southbridges. When set, it indicates that address decode in the host has found no mapping for the given access, and therefore it should be routed to the bus segment containing the southbridge. As part of the initialization sequence, all LDT devices determine whether or not they own (or are) the southbridge. Accesses with the compat bit set are always accepted by devices that own the southbridge, and ignored by all other devices, regardless of address.

4.3.2 Broadcast Message

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]	UnitID[4:0]					
2	Rsv		SrcTag[4:0]/Rsv					
3	Addr[7:2]						Rsv	
4	Addr[15:8]							
5	Addr[23:16]							
6	Addr[31:24]							
7	Addr[39:32]							

Broadcast messages are used by the host to communicate information to all LDT devices. They may only be issued by the host bridge, and travel in the downstream direction the entire length of the chain, being both accepted and forwarded by all devices. Features that are implemented using broadcast messages have reserved address ranges associated with them that are recognized by all devices. All information (including potential write data) necessary to the specific type of operation being performed is carried in the address field.

Broadcast messages have a "response" bit which indicates whether a response should be generated as well as serving as a virtual channel identifier. If the response bit is set the broadcast message travels in the non-posted request channel and responses are generated by all nodes. It is the responsibility of the host bridge to determine how many responses to expect. If the response bit is clear the posted channel is used. For the posted version of the command the SrcTag field is reserved. No assumptions may be made about the uniqueness of SrcTags for posted broadcast messages, either relative to other broadcast messages or other traffic.

4.3.3 Flush

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Rsv			SrcTag[4:0]				
3	Rsv							

Flush is designed to make sure that posted writes have been observed at host memory. It applies only to requests in the same I/O stream as the flush, and may only be issued in the upstream direction. It functions very similarly to a read operation, except that it returns no data, so its Mask/Count and Type fields are reserved. Like reads, it goes in the non-posted request virtual channel. For a flush to perform its intended function, the PassPW bit must be clear, so that the flush pushes all requests in the posted channel ahead of it. It is expected that flushes will never be issued as part of an ordered sequence, so their seqId will always be zero. Flush commands with PassPW set or with a nonzero seqId are legal, but their affect is unpredictable.

Note that flush only guarantees that posted requests have been flushed to their destination within the host. If the requests were peer-to-peer, this only means that they reached their destination host bridge, not final device.

The flush response is returned from the host bridge when the requests have become globally observable in the host. Since there is no data, a TgtDone response is used.

4.3.4 Fence

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Rsv							
3	Rsv							

Fence is designed to provide a barrier between posted writes which applies across all UnitIDs and therefore all I/O streams. It functions only in the upstream direction. It goes in the posted request virtual channel and has no response. There is therefore no SrcTag field in the command packet. A fence with PassPW clear will not pass anything in the posted channel regardless of UnitId. Packets with their PassPW bit clear will not pass a fence regardless of UnitId.

For a fence to perform its intended function, the PassPW bit must be clear so that the fence pushes all requests in the posted channel ahead of it. It is expected that fence commands will never be issued as part of an ordered sequence, so their SeqId will always be zero. Fence commands with PassPW set or with a nonzero SeqId are legal, but their affect is unpredictable.

4.4 Responses (RdResponse and TgtDone)

4.4.1 RdResponse

A node that is the target of read transaction returns a read response packet to the source followed by a data packet which contains the requested data. The format of the read response packet is shown below:

Bit Time	7	6	5	4	3	2	1	0
0	Rsv		Cmd[5:0]					
1	PassPW	Bridge	Rsv	UnitID[4:0]				
2	Count[1:0]		Error	SrcTag[4:0]				
3	Rsv		NXA	Rsv			Count[3:2]	

The Count field encodes the size (in doublewords) minus 1 of the original read request and indicates the size of the data packet, so that intermediate nodes forwarding the response know how much data to expect. For DW packets, the Count is just taken from the request packet. For byte packets, the data field always fits within a single doubleword, so the Count field is always 0 (1 DW).

The Error bit is used to indicate that an error occurred during the read. This could be due to the address accessed being non-existent, or an ECC/Parity error in DRAM or a cache. The requested number of data beats are always driven to the bus, whether they are valid or not, but the Error bit indicates that the data may not be used.

4.4.2 TgtDone

Bit Time	7	6	5	4	3	2	1	0
0	Rsv		Cmd[5:0]					
1	PassPW	Bridge	Rsv	UnitID[4:0]				
2	Rsv		Error	SrcTag[4:0]				
3	Rsv		NXA	Rsv				

TgtDone signals that a transaction has completed at its target. The target may release its buffer as soon as it issues TgtDone. In some cases, TgtDone is not transmitted explicitly, but is implied by RdResponse from a target. TgtDone also has the Error bit, similar to the one in RdResponse, which is used to indicate an error at the target, in those cases where the error is detected prior to sending the response.

4.5 I/O Streams

LDT has the concept of I/O streams, which are groupings of traffic that may be treated independently by the fabric.

Since no peer-to-peer communication exists within the fabric, and all packets travel either to or from the host bridge, the traffic to or from each node in the fabric could in theory be treated independently by the fabric, leaving the host bridge to manage interactions between streams.

Upstream requests contain the ID of the source node, and upstream responses contain the ID of the node that generated the response. Therefore UnitID may be used to identify I/O streams for upstream packets.

Downstream responses contain the ID of the node to which the response is being sent, however downstream requests contain the ID of zero (the encoding reserved for the host bridge), not the ID of the node that is targeted by the request. Therefore it is impossible to determine independent I/O streams in downstream traffic and it must be assumed that all downstream traffic is in the same stream.

The host bridge is responsible for managing interactions between streams. No stream information is propagated through the host bridge. The host bridge is responsible for maintaining ordering within the host domain in whatever fashion is appropriate.

It is permitted for a single physical node to be allocated multiple UnitID's if it generates multiple independent streams of traffic. This will allow more concurrency among the traffic to/from that device. If this is not done, all traffic to/from that device will be ordered as a single stream, and knowledge of the possible concurrency lost.

4.6 Virtual Channels

LDT supports 3 virtual channels of information:

1. Posted Requests
2. Non-Posted Requests (reads, flushes, non-posted writes)
3. Responses

Requests may cause responses to be issued by receiving nodes. Requests received by a host bridge may also cause downstream requests to be issued (peer-to-peer reflection). Non-host bridge nodes may not make accepting a request or issuing a response due to an incoming request dependent on the ability for that node to issue an outgoing request, or receipt of a response due to a request issued by that node.

Responses may cause other responses to be issued. It is the responsibility of nodes generating requests to pre-allocate enough space for all responses (including response data) that may be received, so that responses are always guaranteed to be accepted by their destination node.

All devices must guarantee that the 3 virtual channels are not capable of blocking each other due to buffer management and routing issues, which is why each channel has command and data buffer space separate from the other two. However, in order to properly maintain I/O ordering, some rules are added which create dependencies between packets (in the same I/O stream) in different virtual channels.

BOZO: It would appear that we have a deadlock case here in the double-hosted link case, if the chain is not partitioned between the two host bridges. A deadlocking loop can be formed if peer-to-peer requests are issued by two different intermediate nodes in opposite directions. Each reflected peer-to-peer request coming out of a host bridge can be blocked behind a stack of requests targeting the other host bridge. The host bridge will only be able to queue a finite number of peer-to-peer requests in from the link without issuing one.

4.7 Flow Control

LDT receivers contain the following types of buffers:

1. Non-Posted Commands
2. Posted Commands
3. Responses
4. Non-Posted Command Data
5. Posted Command Data
6. Response Data

Command and Response buffers contain enough storage to store the largest control packet of that type. All data buffers can hold 64 bytes.

The table in section 0 defines the virtual channels and hence the buffers used for each of the control packets.

These buffers are flow controlled at the link level using a coupon-based scheme in which the transmitter contains a counter for each type of buffer at the receiver. At system reset the transmitter clears its counters, and when reset deasserts the receiver sends Nop packets to indicate how many buffers of each type it has

available. When the transmitter sends a non-info packet it decrements the associated counter, and when a particular counter contains a zero the transmitter stops sending packets to the associated buffer. When the receiver frees a buffer it sends a Nop packet to the transmitter, and the transmitter increments the associated counter.

A transmitter may not issue a control packet that has an associated data packet unless the receiver has both the appropriate control and data buffers available. If this is rule were violated one virtual channel may block another and lead to deadlock, since only one data transfer can be pending on the link at a time.

The format of the Nop packet is shown below. Bit 7 of bit time 2 is defined to always be 0.

Bit Time	7	6	5	4	3	2	1	0
0	Rsv		Cmd[5:0]					
1	ResponseData[1:0]		Response[1:0]		PostData[1:0]		PostCmd[1:0]	
2	Rsv	Diag	Rsv		NonPostData[1:0]		NonPostCmd[1:0]	
3	Rsv							

Each 2-bit field in the packet indicates how many buffers of each type have become available. Hence each two-bit field can free zero, one, two or three buffers. Receivers are not limited to having three buffers of a particular type, and can free up additional buffers by sending additional Nop packets.

While the goal is to size each buffer at the receiver to bury the round trip latency from the transmitted packet to the returning Nop packet, this is not strictly required by this specification. It is the responsibility of each device to guarantee that Nop packets cannot be prevented from being issued due to transmission of other traffic, to avoid starvation of the far transmitter.

If a transmitter receives more increments than it can keep track of, it must not allow its counter to wrap, but must discard the extras. This has the effect that the link will use the maximum number of buffers that both the transmitter and receiver can support. All transmitter counters must be a minimum of 4 bits, allowing up to 15 buffers to be tracked without loss.

The Diag bit is used to indicate the beginning of a CRC testing phase, as described in section 11.1.1. Everything following the Nop packet, until the conclusion of the current CRC test interval, will be ignored.

4.8 Routing

LDT has both directed and broadcast traffic. They are not associated with any particular virtual channel, just specific packets. The broadcast request packet travels in the posted command channel. All other packet types are directed. Directed packets are relayed down the fabric until they reach their destination, where they are absorbed. Broadcast packets are relayed down the entire length of the fabric, but also accepted at each node they pass through, and terminated by the node at the far end of the fabric. They may only be initiated by a host bridge.

4.8.1 Acceptance

A node will accept an incoming packet if any of the following are true:

1. The packet is a broadcast request.
2. The packet is a directed request with a UnitID of 0 (indicating it is from a host bridge) and (for packets with a Compat bit) the Compat bit clear, to an address owned by this node.
3. The packet is a directed request with a UnitID of 0 and a set Compat bit, and this node either is the southbridge, or a bridge to the southbridge.
4. The packet is a response with the bridge bit set (indicating it is from a host bridge) and a UnitID owned by this node.

4.8.2 Forwarding

Whenever a node forwards a packet, it always sends it along in the direction it was travelling.

A node will forward an incoming packet to its outgoing link if any of the following are true:

1. The packet is a broadcast request.
2. The packet is a directed request with a UnitID of 0 (indicating it is from a host bridge) and (for packets with a Compat bit) the Compat bit clear, to an address not owned by this node.
3. The packet is a directed request with a UnitID of 0 and a set Compat bit, and this node is neither the southbridge nor a bridge to the southbridge.
4. The packet is a directed request with a non-zero UnitID (indicating that it is from an interior node).
5. The packet is a response with the bridge bit set (indicating it is from a host bridge) and a UnitID that doesn't match this node.
6. The packet is a response with the bridge bit clear (indicating it is from an interior node).

If the device is at the end of the fabric, that fact will be indicated by the EndOfChain CSR bit. In that case, it will be unable to forward packets. If a packet is received that would normally be forwarded, one of the following actions is taken instead, depending on the type of packet received:

1. Broadcast requests are silently dropped; they have successfully traversed the whole fabric.
2. Non-posted directed requests are responded to with a TgtDone (for Writes) or Read Response (for Reads) packet with the error bit set, the NXA bit set, the bridge bit clear, and a unitId of 0. Read responses return the requested number of dwords, with a data value of all 1's.
3. Response and posted request packets are dropped. The node may choose to log this error and report it via a sideband method.

4.8.3 Host Bridges

Host bridges are always at the ends of the fabric, and therefore never forward packets. However, the acceptance of a packet by a host bridge will likely result in action within the host.

Host bridges will take the following action upon receiving a packet:

1. Broadcast requests are silently dropped; they have successfully traversed the whole fabric.
2. Directed requests with a UnitID of 0 are from a bridge on the far end of the fabric. Type 0 configuration accesses to Device Number 0 are directed to the bridge's CSRs. Optionally, the host bridge could also implement a memory or I/O space region addressable from the far host bridge, to be used for messaging in clustered systems. A description of how this would be used and what it would look like is beyond the scope of this specification. In that case, the bridge would respond to accesses to this area as if it were an interior node. The responses would have the bridge bit clear and a UnitID of 0. All requests to addresses not included above are considered accesses to nonexistent addresses. If non-posted, they are responded to with a Target Done (for Writes) or Read Response (for Reads) packet, with the error bit set, NXA bit set, bridge bit clear, and a UnitID of 0. Read responses return the requested number of dwords, with a data value of all 1's. Posted requests to nonexistent addresses generate no response, and are silently dropped. This may be reported as an error via a sideband mechanism.
3. Directed requests with a nonzero UnitID are from interior nodes, and are accepted and handled by the node logic.
4. Responses with the bridge bit set are silently dropped. This means that a host bridge tried to respond to an interior node which did not pick up the response. The node may choose to log this error and report it via a sideband mechanism.
5. Response packets with the bridge bit clear are responses to requests issued by this bridge. The host bridge will match this to one of its outstanding requests. If no match exists the node may choose to log this error and report it via a sideband mechanism.

4.8.4 Fairness and Forward Progress

In order to issue packets, a node must insert them into the stream of traffic that it is forwarding. A node must guarantee that forward progress is always made by not allowing forwarded and injected traffic to

starve each other. A node must also implement a method of assuring fair access to fabric for all nodes, approximating the round-robin behavior of a fair bus.

4.8.4.1 Policy

Each node is allowed to insert packets into a busy link at a rate matching that of the heaviest node forwarding traffic through it. In addition, the node may freely use any idle time on the link. This property must be met over a window in time small enough to be responsive to the dynamic traffic patterns, yet large enough to be statistically convergent. In order for a system of nodes to behave consistently, each node should implement this policy using the same algorithm as described below.

4.8.4.2 Algorithm

The algorithm consists of two parts. The first is the method used to calculate the insertion rate the node can use. The second is governs how the node achieves that insertion rate. This algorithm must be implemented independently for both the upstream and downstream direction to support dual host bridge configurations. The algorithm requires no CSRS and has no configurable parameters.

To calculate the insertion rate, the maximum packet rate must be deduced for the heaviest downstream node. This is done by implementing 32 three bit counters, one for each potential downstream node as well as a single eight bit counter. At reset all counters are reset to 0. When a request is forwarded the three bit counter corresponding to the node that generated the packet is incremented. The eight bit counter is incremented once for every forwarded packet. When one of the three bit counters overflows, the value of the eight bit counter (post increment) is captured (hereafter referred to as the denominator). All counters are then cleared. The request rate of the worst case downstream node has now been calculated and is equal to 8/denominator. The node may, on the average, insert 8 requests for every 'denominator' request forwarded. This insertion should be paced and not inserted as bursts. Packets may always be inserted when there are no packets waiting to be forwarded. The denominator register is set to 1 upon reset.

To insert, the node has an 8 bit counter referred to as 'window', which at reset is set to 1. It also has a one bit register, referred to as priority, that at reset is cleared to 0. When a node has requests ready to be sent on the outbound links, it decides which to send based on the following cases:

1. Forward packet to send; no local packet to send:
   The forward packet is sent and the window register is decrement.

2. No forward packet to send; local packet to send:
   The local packet is sent and the priority register is cleared.

3. Both forward packet and local packet to send:
   If the bit in the priority register is set, the local packet is sent and the priority bit is cleared. Otherwise, the forward packet is sent and the window register is decremented.

Whenever the window register is decremented to 0, its next value is recalculated and the priority bit is set. In order to achieve non-integral insertion rates the new value of the window register must be loaded probabilistically. Each node will implement a 9 bit LFSR using the polynomial x^9+x^4+1. It is advanced once every time the window register value is recalculated. The window register is loaded with (denominator+LFSR[2:0]) >> 3.

4.8.4.3 Implementation Note

Care must be taken in implementing the packet insertion logic in order to avoid a potential starvation problem. The packet inserter is basically a 2-input arbiter between issued packets and forwarded packets. The requests to this arbiter are generated when there is a packet ready to go from one of these sources, and there are free buffers (as indicated by buffer release messages) at the other end of the link to receive the packet. It is possible that there is one packet to be issued and forwarded, both in the same virtual channel, and therefore requiring the same buffer type(s). If the forwarded packet is chosen, and there is only one

buffer of the needed type free, the issued packet will be unable to be transmitted. When the fairness logic next allows a packet to be inserted, a packet from a different virtual channel may be chosen, allowing the priority of the packet inserter to swing back to forwarding. When the buffer release message that would allow the blocked packet to go arrives, it no longer has priority in the inserter, and thus can't go. If another packet in the same channel is forwarded before priority changes back to inserting, this situation can persist, starving packet insertion in a particular virtual channel.

One solution to this problem is to allocate buffers at the receiver to packets before they are allowed to arbitrate at the packet inserter. That way, the buffer cannot be lost while the packet is blocked. If there is only one buffer left, it should be assigned to the oldest packet (either issued or forwarded) in that channel. Note that buffers cannot be statically allocated between packet issuing and forwarding, as the receiver may only implement one buffer of a particular type.

5 LDT I/O Ordering

This section explains the ordering rules for all 3 types of I/O traffic: peer-to-peer, DMA, and PIO. Peer-to-Peer traffic is traffic that has both its requester and target on the LDT I/O link. Since LDT does not support peer-to-peer traffic directly, all peer-to-peer traffic (whether to the same fabric or not) goes upstream into the host and then back downstream. For purposes of ordering, we shall consider the upstream and downstream legs independently. DMA and PIO may be seen as special cases of peer-to-peer I/O traffic that have one end in the host, and skip either the upstream or downstream leg, respectively.

These ordering rules only apply to the order in which operations are seen by targets at the same level of the fabric hierarchy. Consider two ordered peer-to-peer write requests issued by an LDT I/O device to two different targets on different LDT I/O segments. The ordering rules on the originating LDT chain will guarantee that the two writes reach the host bridge in the appropriate order. The host is responsible for guaranteeing that the two writes reach their target host bridges in the correct order. However, beyond that point, the writes are in independent fabrics, and there is no guarantee about the order in which they will reach their final destination. If an I/O device requires assurance of final completion, it must have a way of polling the destination device to determine that the first write has been observed before issuing the second one, or use non-posted writes.

Ordered operations which return responses (reads or non-posted writes) are guaranteed to complete at the target in the correct order, but no guarantee is made about the order in which the returning responses will be received. All LDT I/O devices must be able to accept responses out of order or restrict themselves to one outstanding non-posted request. A bridge that is between LDT and an I/O protocol that requires responses to be returned in order must provide sufficient buffering to be able reorder as many responses as it may have outstanding requests.

5.1 Upstream LDT I/O Ordering

LDT recognizes three types of traffic: posted requests, non-posted requests, and responses, each in a separate virtual channel. These three types of traffic may be distinguished by their LDT command encodings. Requests have a sequence ID (seqId) tag; requests in the same I/O stream and virtual channel with matching non-zero seqId's are considered part of a strongly ordered sequence; sequences are designed to support groups of LDT transactions generated by a single request on the source I/O bus. Requests and responses both have a "may pass posted writes" (PassPW) bit.

For definitions of I/O streams and virtual channels, see sections 4.5 and 4.6, respectively.

LDT has the following upstream ordering rules:

1. Packets from different sources are in independent I/O streams and have no ordering guarantees. Devices receiving packets in different I/O streams may reorder them freely.
2. Packets in the same I/O stream and virtual channel that are part of a sequence (having matching nonzero seqID's) are strongly ordered, and may not pass each other. Devices receiving them must keep them strongly ordered.

3. Packets in the same I/O stream, but not in the same virtual channel or part of the same ordered sequence, have their passing rules given by the following table:

Row pass Column?	Posted Request	Non-Posted Request	Response
Posted Request	PassPW: Yes/No !PassPW: No	Yes	Yes
Non-Posted Request	PassPW: Yes/No !PassPW: No	Yes/No	Yes/No
Response	PassPW: Yes/No !PassPW: No	Yes	Yes/No

No – indicates the subsequently issued transaction is not allowed to complete before the previous transaction to preserve ordering in the system. This implies an interaction between the otherwise independent virtual channels within LDT.

Yes – indicates the subsequently issued transaction must be able to pass the previous transaction, or deadlock may occur. This mean that the packet type given in the column may not be permitted to block the packet type given in the row at any point in the LDT fabric or host.

Yes/No – indicates the subsequently issued transaction may optionally be allowed to complete before the previous transaction if there's any advantage to doing so; there are no ordering requirements between the two transactions. However, support for reordering is not required; failure to reorder the packets will not lead to deadlock.

5.2 Host Ordering Requirements

The host bridge and host system are required to implement the same virtual channels as provided in the LDT I/O fabric, and guarantee that transactions that are ordered within the LDT fabric are ordered within the host. Specifically, this means that the first operation in an ordered pair must be guaranteed visible to all observers within the host domain before the second operation. However, in the case of posted peer-to-peer I/O operations, the host can only guarantee that the first operation has been issued on its destination link; it has no way of knowing whether the operation has reached its final target device.

5.3 Downstream LDT I/O Ordering

The rules for downstream ordering are the same as those for upstream ordering, with the exception that I/O streams are identified by the target of the transaction, rather than the source. The same three virtual channels exist. However, UnitID may not be used to identify unique I/O streams in the downstream direction; so, it must be assumed that all downstream traffic is in the same stream. This asymmetry in the definition of I/O streams for upstream and downstream traffic is why it is important for a node to be able to differentiate upstream responses from downstream responses, and provides the motivation for the **bridge** bit in the response packet.

The host must also guarantee that peer-to-peer LDT traffic that was part of an ordered sequence when received is also emitted downstream as an ordered sequence.

6 Isochronous Commands

BOZO: add blurb that describes isochronous commands and how they are used to support isochronous streams.

Issues:

- **Does the incremental performance gain of dynamic priority modes (ISO vs ASY) justify the added complexity? We assume not for now.**
- **Assuming no dynamic priority (ISO vs ASY) do any systems need dedicated high priority buffers. Assume not.**

7 Interrupts

7.1 Interrupt Requests

All interrupt requests, regardless of interrupt class, are sent from the interrupting device to the host bridge using posted byte WrSized packets to the reserved range defined in section 10. The count field is always 0, which indicates that only a single DW data packet follows the write, and it contains byte masks, not data. The byte masks are not used by the interrupt request and must always be all 0.The table below shows the format of interrupt request packets.

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]		UnitID[4:0]				
2	Count[1:0]		Rsv	SrcTag[4:0]				
3	Rsv	DM	TM	MT[2:0]			Count[3:2]	
4	IntrDest[7:0]							
5	Vector[7:0]							
6	Addr[31:24]							
7	Addr[39:32]							

The host bridge is then responsible for delivery to the correct internal target.

Vector and TM are not used (reserved) for SMI, NMI, STARTUP, and INIT messages.

Interrupt request packets push posted writes if the PassPW bit is clear. All preceding posted writes will be visible at their targets within the host before the interrupt is delivered.

There are three classes of interrupts supported in LDT:
- Arbitrated (Low Priority)
- Fixed
- Non-vectored

Arbitrated interrupts will only be delivered to one of the addressed destinations within the host targeted by the interrupt. The ultimate target will be either the lowest priority destination or a destination that is already servicing the same interrupt source (the Focus Processor). Arbitrated interrupts have 256 possible sources. Each interrupt source is identified by a 8-bit vector ID.

Fixed interrupts are delivered to all destinations addressed by the interrupt message. They can be used to send single, multicast or broadcast interrupts. Fixed interrupts also have 256 possible sources, identified by vector ID.

Non-vectored interrupts carry no source information. They are always delivered to all addressed destinations. They consist of the following types:
- SMI
- NMI
- INIT
- Startup
- Ext Int

The type of interrupt is identified by the Message Type (MT) field, encoded as follows:

MT[2:0]	Message Type
000	Fixed

MT[2:0]	Message Type
001	Lowest Priority Mode
010	SMI
011	NMI
100	INIT
101	Startup
110	Ext Int
111	*Reserved*

Arbitrated and fixed interrupts may either be edge triggered or level sensitive, as identified by the Trigger Mode (TM) field. Edge and level sensitive interrupts cannot be mapped to the same vector. Level sensitive interrupts require an End Of Interrupt (EOI) message to be transmitted to acknowledge the servicing of the interrupt. A subsequent level sensitive interrupt using the same vector will not be sent until an EOI message has been received. Edge triggered interrupts do not signal the servicing of the interrupt. Non vectored interrupts are always edge triggered.

TM is encoded as follows:

TM	Trigger Mode
0	Edge
1	Level

For all three classes of interrupts, the set of potential destinations is determined by the IntrDest and Destination Mode (DM) fields. The DM field determines if IntrDest represents a physical identifier or a logical identifier.

DM	Destination Mode
0	Physical
1	Logical

In physical mode, each interrupt destination (CPU) within the host is assigned a unique 8 bit physical ID. The physical ID 0xFF is reserved and is used to indicate that the interrupt should be broadcast to all possible destinations. A destination will be considered a target for a physical mode interrupt if its ID matches the IntrDest field or if the IntrDest field equals 0xFF.

For logical addressing each interrupt destination is assigned an 8 bit logical ID. The determination of what constitutes a valid logical ID is system-specific, and the method of comparison of logical ID to IntrDest is programmable. As an example, a system may choose a one-hot address representation, assigning one bit to each processor (limited to 8 CPUs), or it may define a portion of the logical address to be fully decoded and the rest of the bits to be one-hot encoded.

LDT will never collapse interrupts.

7.2 EOI

EOI messages are sent in posted Broadcast Message packets to all nodes across the LDT I/O fabric. Each device is responsible both for accepting the EOI and clearing outstanding interrupts to the specified vector ID, and passing the EOI down the fabric. The format of an EOI packet is shown below

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]	UnitID[4:0]					
2	Rsv		SrcTag[4:0]					
3	Rsv		MT[2:0]				Rsv	

Bit Time	7	6	5	4	3	2	1	0
4	Rsv							
5	Vector[7:0]							
6	Addr[31:24]							
7	Addr[39:32]							

EOIs use the same reserved address range as interrupt requests. The MT field must always contain the value 111 (EOI). Vector contains the interrupt vector of the interrupt to be acknowledged.

7.3 Interrupt Acknowledge

An interrupt acknowledge transaction may be directed to the interrupt controller by performing a byte read within the reserved IACK range defined in section 10. Any read within this address range will generate a RdSized command with the compat bit set. This command packet will be routed directly to the southbridge if the southbridge is a native LDT device. If the southbridge is implemented as a PCI device then the command packet will be routed to the intervening LDT-PCI bridge. The bridge will generate an interrupt acknowledge cycle on the PCI. In both cases the read response will contain the interrupt vector which will be in the least significant byte lane position independent of the byte masks in the RdSized command.

8 Configuration Accesses

LDT implements configuration space similarly to PCI, as defined in the PCI Local Bus Specification, rev 2.2. LDT devices and bridges (including host bridges) must implement appropriate PCI configuration headers. Busses and devices are numbered in a fashion that maps into PCI bus and device numbers. Configuration software should be able to do configuration across LDT in a way that is indistinguishable from an equivalent PCI bus hierarchy. Configuration space is mapped to a predefined region of the LDT system address space. See section 10 for details.

8.1 Configuration Cycle Types

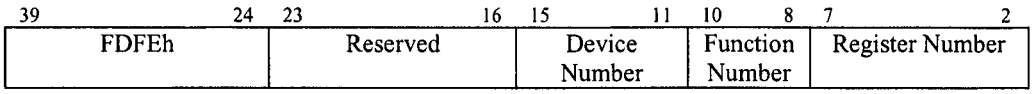
PCI uses two types of configuration cycles, type 0 and type 1. The two types are needed because it must be possible to access the configuration space of devices on a bus without the devices knowing what bus they're on.

Type 0 cycles are used to access devices on the current bus. They contain a function number and register number. The bus number is implicitly the current bus. The device number is indicated by the idsel# pins, which are asserted as appropriate by the bridge. Thus, PCI devices do not need to know their bus number or device number in order to respond to configuration accesses.

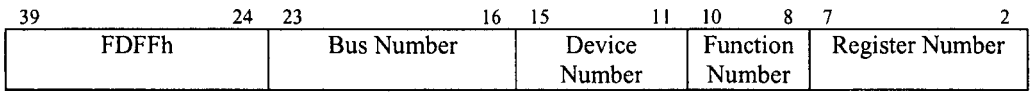
Type 1 cycles are used to transmit configuration cycles over intermediate busses, and contain the bus number, device number, function number and register number fields. Bridges forward type 1 cycles through the bus hierarchy, and translate them to type 0 cycles when driving them onto their final destination bus. Host bridges may optionally implement the capability to transmit PCI Special Cycles to remote busses using device 31, register 0, type 1 configuration cycles.

LDT also requires two types of configuration cycles, for the same reasons.

An LDT Type 0 access is performed by issuing a RdSized or nonposted WrSized command with an address of the following form. They are only issued by host bridges, and thus always travel downstream. Unlike PCI, LDT Type 0 accesses contain the device number, because all LDT devices know what their device number is. LDT has no analog of the idsel# lines. Host bridges that support double-ended links will respond to type 0 accesses on their secondary interface as Device Number 00h. Note that in a double-hosted link, this implies that both bridges are responding to the same address; which one you are talking to is determined by which direction the packets are travelling in. This function is only intended to be used by system sizing firmware.



An LDT Type 1 access is performed by issuing a RdSized or nonposted WrSized command with an address of the following form. In general, type 1 accesses are issued by host bridges. The only type 1 accesses that can flow upstream from a slave device are Special Cycle requests. The system's response to all other upstream type 1 requests is undefined.



RdSized and WrSized configuration accesses of greater than 1 dword are not supported.

8.2 Configuration Space Mapping

8.2.1 Function and Register Numbering

The numbering of functions and registers within a device is device-specific, except that every implemented device number must have a function 0 that contains a standard PCI configuration header, identifying the device. Certain other standard CSRs are required by the LDT specification.

8.2.2 Device Numbering

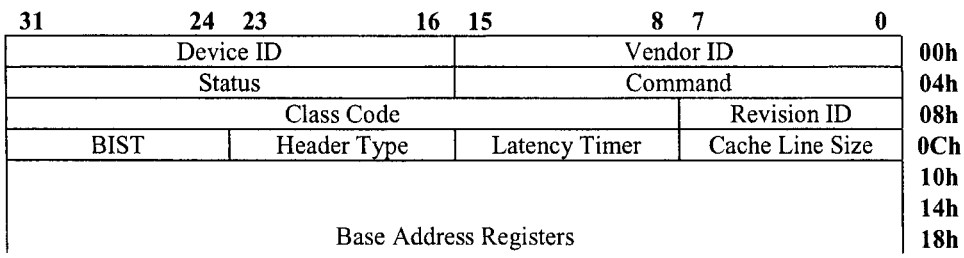
Devices in LDT are identified by UnitIDs, which range from 1-1Fh. A single physical device may own multiple UnitID values. Every LDT device owns the device numbers that correspond to its UnitIDs. It must implement a CSR space (with configuration header) at the Device number equal to its lowest UnitID value. It may choose to implement CSR spaces corresponding to any number of its remaining UnitIDs, including none. Each implemented space must contain an appropriate PCI header. Unimplemented spaces must not be responded to by the device. Accesses to these device numbers will not be accepted by any device on the LDT chain, and so will receive a response with the error and NXA bits set.

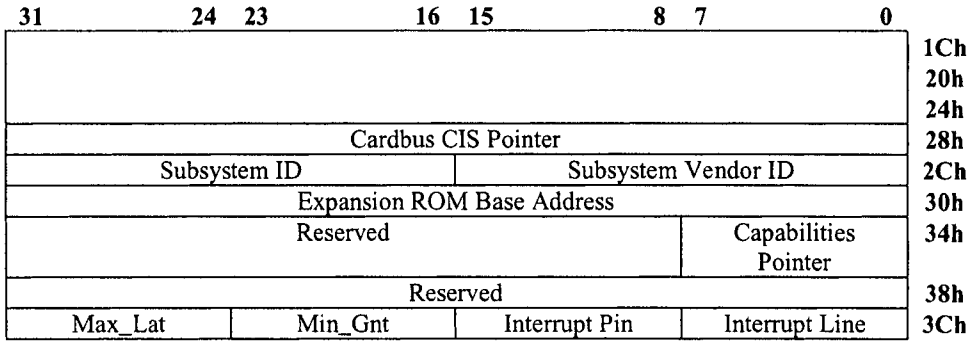
8.2.3 Bus Numbering

Each LDT chain in the system is assigned a single bus number. For double-hosted physical chains, which are logically partitioned between two host bridges, each logical chain has a separate bus number.
Bus numbers are assigned by system initialization software at reset, and follow the conventions used for PCI bus numbering, which require that the bus tree be numbered in a depth-first fashion. No distinction is made between PCI busses and LDT chains in the numbering.

8.3 LDT Device Header

Devices that sit on LDT that perform non-bridging functions implement PCI Device Headers, as shown below. The fact that this is a Device Header is contained in the Header Type register. Connecting to two links in an LDT chain does not constitute a bridging function. Fields in an LDT Device Header are the same as defined in the PCI Local Bus Specification (rev 2.2), with the exceptions listed in the sections below.





Some fields in the headers and LDT capability blocks are defined to be ***persistent*** through warm reset, which means their values are only affected by cold (power-on) reset. If not specified otherwise, a field is non-persistent.

8.3.1 Command Register

The following bits are implemented in an LDT device's command register. All other bits are not applicable to LDT, and must be hardwired to 0.

8.3.1.1 I/O Space Enable (bit 0)

This bit must be set for the device to accept any non-compatibility accesses to the PCI I/O Address Space, as given in section 10. If this device is a subtractive decode device, requests with the Compat bit set will still be accepted, regardless of the state of this bit.

8.3.1.2 Memory Space Enable (bit 1)

This bit must be set for the device to accept any non-compatibility accesses to the Memory Mapped I/O Address Space, as given in section 10. If this device is a subtractive decode device, requests with the Compat bit set will still be accepted, regardless of the state of this bit.

8.3.1.3 Bus Master Enable (bit 2)

This bit must be set to allow the device to issue requests onto the LDT chain. Configuration, system management and interrupt requests are issued irrespective of the state of this bit. All other requests require this bit to be set in order to be issued. Requests from other devices on the chain may still be forwarded, independent of the state of this bit.

8.3.1.4 SERR# Enable (bit 8)

If set, the device will flood all its outgoing links with sync packets in the event that it detects a fatal error. If clear, the device may not generate sync packets except as part of initial link synchronization, although it may still propagate them from one link to the other. This bit is read/writable by software, and resets to 0.

8.3.2 Status Register

The following bits are implemented in an LDT device's status register. All other bits are not applicable to LDT, and must be hardwired to 0.

8.3.2.1 Capabilities List (bit 4)

This read-only bit will always be set to 1, to indicate that the device has a capabilities list containing (at least) LDT-specific configuration information.

8.3.2.2 Signaled Target Abort (bit 11)

This bit is set by an LDT device that returns an error response to a transaction addressed to it. This bit does not get set by an EndOfChain device when it returns an NXA error response due to a request not being accepted by any devices. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It may be cleared by software by writing a 1 to it.

8.3.2.3 Received Target Abort (bit 12)

This bit is set by an LDT device that receives an error response (other than an NXA response) to a request it issued. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It may be cleared by software writing a 1 to it.

8.3.2.4 Received Master Abort (bit 13)

This bit is set by an LDT device that receives an NXA error response to a request it issued. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It may be cleared by software writing a 1 to it.

8.3.2.5 Signaled System Error (bit 14)

This bit is set by an LDT device that has flooded the link with sync packets to signal a system error. This bit is persistent through warm reset. The state of this bit after cold reset is 0. It may be cleared by software by writing a 1 to it; however, software will not be able to access the device via the flooded link.

8.3.3 Cache Line Size Register

This register is not implemented by LDT devices.

8.3.4 Latency Timer Register

This register is not implemented by LDT devices.

8.3.5 Base Address Registers (BARs)

Base address registers for LDT devices are implemented as described in the PCI Spec. Incoming addresses in the Memory Mapped I/O Space address range (as described in section 10) are compared directly to the values in memory space BARs. Incoming addresses in the PCI I/O Space address range (which is only a 25 bit space) have only their bottom 25 bits compared to the I/O space BARs. Bits 31:26 of the I/O space BAR must be 0 for a match to occur.

8.3.6 CardBus CIS Pointer

This register is not implemented by LDT devices.

8.3.7 Capabilities Pointer

Every LDT device will have a capabilities pointer to a linked capabilities list that contains (at least) the LDT-specific capability registers.

8.3.8 Interrupt Line, Interrupt Pin, Min_Gnt, and Max_Lat Registers

These registers are not implemented by LDT devices, and will return 0's if read.

8.4 LDT Bridge Headers

Devices that bridge between LDT and other bus protocols (including subsidiary LDT chains) implement a PCI bridge header as described in the PCI-to-PCI Bridge Architecture Specification (rev 1.1), with the exceptions listed below. Note that LDT may be the primary or secondary bus of the device, or both. Some register bits have LDT-specific meanings only when LDT is connected to a specific port of the bridge. If that interface is to a non-LDT bus, the requirements of that bus protocol determine the meaning of the bit.

31 24	23 16	15 8	7 0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Primary Latency Timer	Cache Line Size	0Ch
Base Address Register 0				10h
Base Address Register 1				14h
Secondary Latency Timer	Subordinate Bus Number	Secondary Bus Number	Primary Bus Number	18h
Secondary Status		I/O Limit	I/O Base	1Ch
Memory Limit		Memory Base		20h
Prefetchable Memory Limit		Prefetchable Memory Base		24h
Prefetchable Base Upper 32 Bits				28h
Prefetchable Limit Upper 32 Bits				2Ch
I/O Limit Upper 16 Bits		I/O Base Upper 16 Bits		30h
Reserved			Capabilities Pointer	34h
Expansion ROM Base Address				38h
Bridge Control		Interrupt Pin	Interrupt Line	3Ch

8.4.1 Command Register

All of the command register bits implemented in the Device Header Command Register (section 8.3.1), are implemented in bridges that have LDT on their primary bus, and affect operation only on the primary bus. If the Bus Master Enable bit is cleared in a bridge device, preventing forwarding of transactions to the primary bus, transactions that would be forwarded must be master-aborted on the secondary bus. If the secondary bus is LDT, this is indicated by returning a response with the NXA bit set.

In addition, bridge devices have the option of implementing the VGA Palette Snoop Enable bit (bit 5). If not implemented, it is hardwired to 0. This bit functions similarly to the way it is described in the PCI Bridge Architecture Spec. If the bit is set, WrSized operations to addresses in the first 64 KB of the PCI I/O Address Range (as defined in section 10) have address bits 9:0 compared to addresses 3C6h, 3C8h, and 3C9h. (Address bits 1:0 are not included in WrSized packets, and so must be determined from the address of the least significant enabled byte in the packet.) Accesses on a primary LDT bus that match will be forwarded to the secondary bus of the bridge. Accesses on a secondary LDT bus that match will not be accepted, regardless of the state of other address decode registers.

8.4.2 Status, Cache Line Size, Primary Latency Timer, Base Address, Interrupt Pin, and Interrupt Line Registers

For a bridge with LDT as its primary bus, these registers are implemented the same way as the corresponding registers in an LDT Device Header. They are not related to the secondary bus.

8.4.3 Secondary Latency Timer Register

If the secondary bus is LDT, this register is reserved.

8.4.4 Secondary Status Register

If the secondary bus is LDT, most of the bits defined in the PCI to PCI Bridge Architecture Spec are not relevant, and are reserved, being hardwired to 0. The exceptions are listed below:

8.4.4.1 Signaled Target-Abort (bit 11)

If set, this bit indicates that the bridge has issued a non-NXA error response on the secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and may be cleared by software by writing a 1 to it.

8.4.4.2 Received Target-Abort (bit 12)

If set, this bit indicates that the bridge has received a non-NXA error response from the secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and may be cleared by software by writing a 1 to it.

8.4.4.3 Received Master-Abort (bit 13)

If set, this bit indicates that the bridge has received an NXA error response from the secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and may be cleared by software writing a 1 to it.

8.4.4.4 Detected System Error (bit 14)

If set, this bit indicates that the bridge detected sync packet flooding on its secondary bus. This bit is persistent through warm reset. It is cold reset to 0, and may be cleared by software writing a 1 to it.

8.4.5 Memory and Prefetchable Memory Base and Limit Registers

For accesses coming in on an LDT bus, these registers are only compared to addresses in the Memory Mapped I/O range, as defined in section 10. Matching addresses are forwarded from the primary to the secondary bus, and ignored on the secondary bus. Non-matching addresses are forwarded from the secondary to the primary bus, and ignored on the primary bus.

8.4.6 I/O Base and Limit Registers

For accesses coming in on the primary (LDT) bus, these registers are only compared to addresses in the 32 MB PCI I/O range, as defined in section 10. Only the low 25 bits of the incoming byte address are used. All bits above bit 24 are forced to 0 before the comparison. Matching addresses are forwarded from the primary to the secondary bus, and ignored on the secondary bus. Non-matching addresses are forwarded from the secondary to the primary bus, and ignored on the primary bus.

8.4.7 Capabilities Pointer Register

Every bridge with LDT on at least one port will have a capabilities pointer to a linked capabilities list that contains (at least) the LDT-specific capability registers.

8.4.8 Bridge Control Register

All unspecified bits are reserved, and return 0 if read.

8.4.8.1 Parity Error Response Enable (bit 0)

If the secondary bus is LDT, this bit is reserved.

8.4.8.2 SERR# Enable (bit 1)

This bit controls the mapping of system errors from the secondary to the primary bus of the bridge. If set, and the SERR# Enable in the Command register is set, system errors will propagate. System errors in LDT are indicated by flooding the chain with sync packets.

8.4.8.3 ISA Enable (bit 2)

This bit is implemented similarly to the way it is described in the PCI Bridge Architecture Spec. For LDT requests in the bottom 64 KB of PCI I/O space (see section 10), this modifies the response to accesses that hit in the range defined by the I/O Base and Limit registers.

8.4.8.4 VGA Enable (bit 3)

This bit functions similarly to the way it is described in the PCI Bridge Architecture Spec. If enabled, RdSized and WrSized operations within the address range 0_000A_0000 to 0_000B_FFFF (inclusive) or within the first 64 KB of the PCI I/O range (as defined in section 10), with address bits 9:0 in the (inclusive) range 3B0h – 3BBh or 3C0h – 3DFh, are forwarded from the primary to the secondary interface, and ignored on the secondary interface, overriding the values in the Memory Base and Limit and I/O Base and Limit registers.

8.4.8.5 Master-Abort Mode (bit 5)

This bit controls the behavior on the source bus when a transaction forwarded through the bus receives a master abort on the target bus. In LDT, master aborts are indicated by a request receiving NXA error response.

If the Master-Abort Mode bit is set, and a nonposted request forwarded from an LDT chain receives a master abort on the target bus, the LDT request will receive an error response with the NXA bit clear. (That is, it will be signaled as an internal error on the originating chain.) If the request was posted, it is not possible to return an error response. This can be signaled by a sideband method.

If the Master-Abort Mode bit is clear, the request will appear to complete normally on the originating LDT chain. Writes will receive TargetDone. Reads will receive ReadResponses, with the appropriate amount of data, which will be all Fs.

8.4.8.6 Secondary Bus Reset (bit 6)

Implementation of this bit is optional. If the secondary bus of the bridge is an LDT chain, writing a 1 to this bit will cause the Reset_L line for that chain to be asserted. If the Warm Reset bit in the Host/Secondary Interface Command Register (section 8.5.3.3.1) is clear, the PwrOk line for that chain will also be deasserted. When, once set, the Secondary Bus Reset bit is cleared, the chain will come out of reset. If the Warm Reset bit is set, this simply results in the deassertion of Reset_L. It is the responsibility of software to delay deasserting the reset long enough to satisfy Reset_L's pulse width requirement. If the Warm Reset bit is clear, clearing the Secondary Bus Reset will cause PwrOk to assert. Hardware will then wait for the appropriate amount of time and deassert the Reset_L pin. If software wishes to be able to determine that the bus has come out of reset, it may poll the Initialization Complete bit of the Link Control Register (section 8.5.3.4.5).

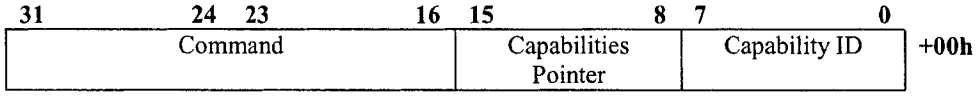
8.4.8.7 Fast Back-to-Back Enable, Primary Discard Timer, Secondary Discard Timer, Discard Timer Status, Discard Timer SERR# Enable (bits 11:7)

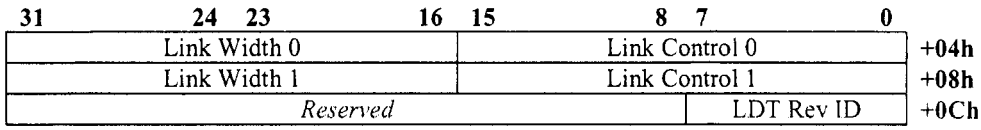
All of these bits are controls for the secondary interface of the bridge. If the secondary interface is LDT, they are reserved, and hardwired to 0.

8.5 LDT Capability Registers

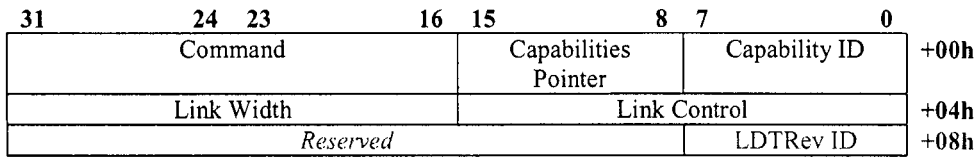
LDT-specific configuration and status information is mapped into configuration space using the Capabilities List methodology described in the PCI Local Bus Specification. A device with multiple LDT interfaces (e.g., an LDT-LDT bridge, with LDT on both the primary and secondary interface) must implement one capability block for each interface. The layout of the capabilities block is determined by the value in the Capability Type field in the command register, but the Capability ID register, Capabilities Pointer register, and Capability Type field are always the same. The offset that the block begins at is implementation specific.

The layout of a Slave/Primary Interface block is as follows:





The layout of a Host/Secondary Interface block is as follows:



8.5.1 Capability ID

This is a read-only register. The capability ID for LDT is 08h.

8.5.2 Capabilities Pointer

This read-only register contains a pointer to the next capability in the list, or a value of 00h if this is the last one.

8.5.3 Command Register

Command[15:0] contains bits used to configure the LDT interface. All unspecified bits are reserved, and are hardwired to 0.

8.5.3.1 Capability Type (bits 15:13)

This field is read-only, and indicates the type of information present in this capability block.

Currently, only two valid encodings are defined. The rest are reserved.

Encoding	Capability Type
000	Slave or Primary Interface
001	Host or Secondary Interface

Primary and Secondary Interface encodings indicate that this block is used to configure the primary (including the interface of an LDT slave device) or secondary (including the interface of a host bridge) interface of a device, respectively.

The layout of the rest of the bits of the Command Register depends on the Capability Type field, as given below.

8.5.3.2 Slave/Primary Interface Command Bits

8.5.3.2.1 Base UnitID (bits 4:0)

This register contains the lowest numbered unitID belonging to this device. If the device owns multiple unitIDs, the additional ones occupy the next consecutive unitID values above the base. The contents of this field are used to generate the unitID field in request and response packets issued by this device, identify responses returning to this device, and identify configuration requests directed to this device. It is readable and writable from software. Its value at reset is 00h.

8.5.3.2.2 Unit Count (bits 9:5)

This read-only field contains the number of unitIDs this device requires. Thus, the highest unitID used by this device is given by (BaseUnitID + UnitCount – 1). If this number exceeds 1Fh, the behavior of the device is undefined.

8.5.3.2.3 Master Host (bit 10)

This bit indicates which link is the path to the master (or only) host bridge on the LDT chain. It is readable from software, but not directly writable. Any time the command register is written, this bit is loaded with the link number that the write came from. Its value at reset is 0.

8.5.3.2.4 Default Direction (bit 11)

This bit determines the default direction for an LDT device to send requests. A 0 indicates requests should be sent toward the master host bridge, as indicated by the Master Host bit. A 1 indicates requests should be sent in the opposite direction. An intelligent device may choice to ignore this if it bases the direction of DMA upon some other criteria. This bit can be read and written by software. It is initialized to 0 at reset.

8.5.3.3 Host/Secondary Interface Command Bits

8.5.3.3.1 Warm Reset (bit 0)

This optional bit allows a reset sequence initiated by Secondary Bus Reset bit of the Bridge Control Register (see section 8.4.8.6) to be either warm or cold. The contents of this bit are only looked at when a software reset sequence is initiated. If it is 0, PwrOK will be driven low as part of the sequence, causing a cold reset. It is the responsibility of the hardware to sequence PwrOK and Reset_L correctly. If this bit is implemented, it is read/writable by software and its value after reset is 1. If not implemented, this bit is read-only, and hardwired to 1. Changing the state of the Warm Reset bit while the Secondary Bus Reset bit is asserted results in undefined behavior.

8.5.3.3.2 Double Ended (bit 1)

This bit indicates that there is another bridge at the far end of the LDT chain. It is readable and writable by software, and resets to 0. For bridges that don't support double-ended chains, this bit may be hardwired to 0.

8.5.3.4 Link Control Register

Host/secondary interface blocks implement one copy of this register; device/primary interface blocks implement 2, one for each link. For devices that only implement one link in the chain, all bits of the second control register will be reserved and hardwired to 0, except for the LinkFail, EndOfChain, and TransmitOff fields, which will be hardwired to 1.

8.5.3.4.1 CRC Flood Enable (bit 1)

If clear, this bit prevents CRC errors from generating sync packets and causing the system to come down, and from setting the LinkFail bit. CRC checking logic will still run on all lanes enabled by LinkWidthIn, however, and detected errors will still set the CRC Error bits. Its reset value is 0.

8.5.3.4.2 CRC Start Test (bit 2)

When this bit is written to a 1 by software, the hardware will initiate a CRC test sequence on the link, as described in section 11.1.1. When the test sequence is complete, and CRC has been checked on all CRC intervals containing test pattern data, hardware will clear the bit. Software may determine that the test has completed by reading the bit, and check the status of the CRC Error bits. Implementation of CRC test pattern generation is optional. If not implemented, this bit must be hardwired to 0. This bit resets to 0.

8.5.3.4.3 CRC Force Error (bit 3)

When this bit is set, bad CRC will be generated on all transmitting lanes, as enabled by LinkWidthOut. The covered data is not affected. It is read/writable from software, and resets to 0.

8.5.3.4.4 Link Failure (bit 4)

The LinkFail bit is set if a failure has been detected on the link. Devices with only one LDT link will hardwire LinkFail for the nonexistent link to 1. This register is persistent through warm reset. If the bit is set, reset hardware will not attempt to synchronize that link; it will continue to drive the link reset value, as defined in section 13.2. This will prevent the Initialization Complete bit from getting set on either end of the link. Implemented LinkFail bits will be reset to 0 on a power-on reset.

LinkFail bits are set by hardware in the event of a CRC error (assuming CRC Flood Enable is set). The bits are also software writable, allowing software to disable links, or re-enable failed links. Clearing a set LinkFail bit has no effect except to allow that link to be initialized on the next warm reset. Setting a clear LinkFail bit, once link initialization is complete, has no effect except to prevent that link from being initialized on the next warm reset.

8.5.3.4.5 Initialization Complete (bit 5)

This read-only bit is reset to 0, and set by hardware when low-level link initialization is successfully complete. If there is no device on the other end of the link, or if that device is unable to properly perform the low-level link initialization protocol, the bit will never get set. Software should not attempt to generate any packets across the link until this bit is set. CRC checking in the hardware will not begin until initialization is complete.

8.5.3.4.6 End of Chain (bit 6)

The EndOfChain bit is set to indicate that the given link is not part of the logical LDT chain. Packets which are issued or forwarded to this link are either dropped or result in an NXA error response, as appropriate (see section 4.8.2). Packets received from this link are ignored, and CRC is not checked.

EndOfChain resets to 0. It may be set by software by writing a 1, to indicate the logical end of the chain, or partition a double-hosted chain into two independent logical chains. This bit may not be cleared by software – a write of 0 to this bit position has no effect.

8.5.3.4.7 Transmitter Off (bit 7)

This bit provides a mechanism to shut off a link transmitter for power savings or EMI reduction. When set, no output signals on the link toggle. This bit resets to 0, and may be set by software writing a 1 to the bit. This bit may not be cleared by software – a write of 0 to this bit position has no effect.

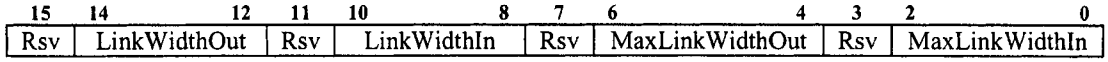
8.5.3.4.8 CRC Error (bits 11:8)

These bits are set by hardware when a CRC error is detected on an incoming link. Errors are detected and reported on a per byte lane basis where bit 8 corresponds to the least significant byte lane. Four bits are required to cover the maximum LDT width of 32 bits. Error bits for unimplemented (as specified by Max Link Width In, section 8.5.3.5.1) or unused (as specified by Link Width In, section 8.5.3.5.3) byte lanes return 0 when read.

These bits are persistent through warm reset, and are reset to 0 at power-on reset. They are readable from software, and may be individually cleared by software by writing a 1.

8.5.3.5 Link Width Register

As with the Link Control Register, there may be either one or two copies of this register, one for each link. If only one link is implemented by the device, the second register is reserved. All unspecified bits are reserved.



8.5.3.5.1 Max Link Width In (bits 2:0)

This field contains 3 bits that indicate the physical width of the incoming side of the LDT link implemented by this device. It is read-only.

The encodings are as follows:

LinkWidth[2:0]	Width
000	8 bit
001	16 bit
010	Reserved
011	32 bit
1xx	Reserved

8.5.3.5.2 Max Link Width Out (bits 6:4)

This field contains 3 bits that indicate the physical width of the outgoing side of the LDT link implemented by this device. It is read-only. It uses the same encodings as the MaxLinkWidthIn field.

8.5.3.5.3 Link Width In (bits 10:8)

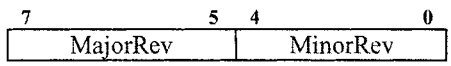
This field controls the utilized width (which may not exceed the physical width) of the incoming side of the links of the LDT link implemented by this device. It uses the same encodings as the MaxLinkWidthIn field of the Control CSR. It is read/writable from software, and persistent through warm reset. At cold reset, this field is reset to 000b (8 bit). Based on sizing the devices at both ends of the link, software may then write a larger value into the register. The chain must pass through warm reset for the new width values to be reflected on the link.

The LinkWidthIn CSR in the link receiver must match the LinkWidthOut CSR in the link transmitter of the device on the other side of the link. The LinkWidthIn and LinkWidthOut registers within the same device are not required to have matching values. The encodings are chosen so that software may read the bottom byte of the LinkWidth CSR in the devices at each end of the link, perform a bitwise AND of them, and write the result into the top byte of the LinkWidth CSR at each end of the link.

8.5.3.5.4 Link Width Out (bits 14:12)

This field is similar to the LinkWidthIn field, except that it controls the utilized width of the outgoing side of the links implemented by this device. Byte lanes that are disabled due to the LinkWidthOut value being set narrower than the physically implemented width of the link will have their transmitters shut down in the same way as if TransmitterOff was set.

8.5.3.6 LDT Revision ID Register



8.5.3.6.1 Minor Revision (bits 4:0)

This read-only field contains the minor revision of the LDT specification that the particular implementation conforms to.

8.5.3.6.2 MajorRevision (bits 7:5)

This read-only field contains the major revision of the LDT specification that the particular implementation conforms to.

9 System Management

LDT system management supports several system-level functions. This section lists each of the functions and the means by which they are implemented using LDT system management messages. Here is a complete list of the types of information flow supported by system management packets.

1. X86 legacy signals: information flows in both directions between the LDT device that has the southbridge and the associated host bridge.
2. Special cycles: information flows downstream from the host bridge at node 0 to the LDT device which has the southbridge.
3. ACPI power management: information flows in both directions between the LDT device that has the system management controller (which is typically located in the southbridge) and the associated host bridge.

9.1 Command Mapping

System management commands are performed by doing posted byte WrSized operations to the system management address range defined in section 10. The count field is always 0, which indicates that only a single DW data packet follows the write, and it contains byte masks, not data. The byte masks are not used by the system management command and must always be all 0. The format these packets is shown below:

Bit Time	7	6	5	4	3	2	1	0
0	SeqId[3:2]		Cmd[5:0]					
1	PassPW	SeqId[1:0]	UnitID[4:0]					
2	Count[1:0]		Rsv	SrcTag[4:0]				
3	Rsv						Count[3:2]	
4	SysMgtCmd[7:0]							
5	Addr[23:20]				Rsv			
6	Addr[31:24]							
7	Addr[39:32]							

The type of system management command (SysMgtCmd[7:0]) is encoded as follows:

SysMgtCmd	Command Type
0000 xxxx	Reserved
0001 xxxx	X86 legacy inputs to CPU. New state of signal: [0]: IGNNE [1]: A20M [2]: STPCLK [3]: Reserved
0010 xxxx	X86 legacy output from CPU. New state of signal: [0]: FERR [1]: SMIACK [3:2]: Reserved
0011 xxxx	Reserved
0100 xxxx	ACPI State Transitions. [3:0] are subcode: 0: Go to C0 1: Go to C1 2: Go to C2 3: Go to C3 4: Go to S0 5: Go to S1 6: Go to S2 7: Go to S3 8: Go to S4

SysMgtCmd	Command Type
	9-15: Reserved
0101 0xxx	Special Cycles and ACPI acknowledgement. [3:0] are subcode: 0: SHUTDOWN 1: HALT 2: STOP-GRANT 3: Reserved 4: ACPI transition ACK – transition complete 5: ACPI transition NACK – request for transition rejected 6 – 15: Reserved
0110 xxxx	Temp High (Temperature increased above threshold). [3:0]: Encode location of temperature sensor
0111 xxxx	Temp OK (Temperature dropped below threshold) [3:0]: Encode location of temperature sensor
1xxx xxxx	Reserved

9.2 X86 Legacy Signals: Inputs to CPU

The information associated with the x86 legacy signals is transported using system management packets in LDT systems. The legacy signals that are inputs to CPUs are:

- IGNNE
- A20M
- STPCLK

These packets originate from the LDT device that has the southbridge and are sent upstream to the associated host bridge, which will cause the correct signals specified in the SysMgtCmd to get set to the given state at the boot strap processor (BSP).

9.3 X86 Legacy Signals: Outputs from CPU

The legacy signals that are outputs from CPUs are:

- FERR
- SMIACK

Whenever the FERR or SMIACK pins change state, the BSP directs a system management packet to the host bridge that owns the southbridge, which in turn sends the system management packet down the LDT link to the southbridge.

9.4 Special Cycles

The special cycles carried by system management packets are:

- HALT
- SHUTDOWN

These transactions are forwarded down the appropriate LDT link to the southbridge. The action taken by the system in response to special cycles is outside the scope of this specification.

9.5 ACPI Power Management

LDT provides a hardware mechanism to sequence ACPI state transitions of the BSP. This hardware mechanism involves information flow between the BSP and the LDT I/O node that has the system management controller (SMC). It is assumed that ACPI state sequencing of other elements of the system is done under software control, and is outside the scope of this specification. The elements of the system that have ACPI state transitions sequenced in an implementation-specific way include:

- Application processors (processors other than the BSP)
- Communication resources between the BSP and application processors.
- LDT links and I/O bridges other than the connection between the SMC and the host.
- Non-LDT bridges, busses or devices

In the following sections, ACPI transitions are described within the context of a multiprocessor system in order to illustrate the principles. In practice many ACPI state transitions only occur in uniprocessor systems.

9.5.1 CPU-level ACPI state transitions

LDT system management packets are defined to support hardware controlled transitions between the following CPU-level ACPI states:

- C0: normal operation
- C2: lower power than C0, snoop traffic still supported. This corresponds to the legacy Stop Grant state.
- C3: lower power than C2, snoop traffic not supported. This corresponds to the legacy Sleep state.

Within this context AUTOHALT would correspond to the C1 state. This is not a hardware-invoked state so is outside the scope of this specification.

The system management controller (SMC) invokes ACPI state transitions that affect the BSP. BSP ACPI state transitions are invoked with LDT system management transactions directed from the SMC node to its associated host bridge, which passes it to the BSP. When the BSP has transitioned to the specified ACPI state, logic associated with that processor generates an explicit positive acknowledgement by directing the appropriate system management transaction to the SMC. The host may also generate a negative acknowledgement, indicating that the ACPI request is being rejected. Only one unacknowledged system management transaction from the SMC may be in progress at a time.

9.5.2 System Level ACPI State Transitions

LDT system management packets are defined to support transitions between the following system-level ACPI states:

- S0: Normal operation
- S1 – S4:

The manner in which these LDT transactions are deployed, and the affect that they have on system operation are beyond the scope of this specification.

9.6 Disconnecting and Reconnecting LDT Links

BOZO: I'm tired, leave this for another day.

10 System Address Map

Base Address	Top Address	Size	Use
00 0000 0000	FC FFFF FFFF	1008 GB	DRAM / Memory Mapped I/O
FD 0000 0000	FD F7FF FFFF	3968 MB	Reserved
FD F800 0000	FD F8FF FFFF	16MB	Interrupts
FD F900 0000	FD F90F FFFF	1MB	IACK
FD F910 0000	FD F91F FFFF	1MB	System Management
FD F920 0000	FD F92F FFFF	1MB	PREQ Protocol (ISA deadlock prevention)
FD F930 0000	FD FBFF FFFF	45MB	Reserved
FD FC00 0000	FD FDFF FFFF	32 MB	PCI I/O

Base Address	Top Address	Size	Use
FD FE00 0000	FD FFFF FFFF	32 MB	PCI Configuration
FE 0000 0000	FF FFFF FFFF	8 GB	Reserved

The bulk of the address space may be used for either memory or memory-mapped I/O. The partitioning of this space into regions for each is implementation-specific.

11 Error Handling

11.1 Error Conditions

BOZO: I'm listing here all the error types that I have seen discussed. I make no guarantees as to completeness.

BOZO: A couple of other potentially interesting error types, not currently covered anywhere:

1. ***Requests that fall off the end of the I/O fabric.***
2. ***Responses delivered to nodes that don't think they issued the corresponding request.***

11.1.1 Transmission Errors

A 32 bit CRC covers all LDT links. The CRC is calculated on each 8 bit lane independently (regardless of whether it is part of a larger link or not), and covers the link as a whole, not individual packets. CTL is included in the CRC calculation. In each bit time, CAD is operated on first, beginning with bit 0, and followed by CTL. For 16 and 32 bit links, where the upper byte lanes do not have a CTL bit associated with them, a CTL value of 0 (Data) is used.

The CRC is computed over 512 bit times. Each new CRC value is stuffed onto the CAD bits of the link 64 bit times after the end of the 512 bit time window, and occupies the link for 4 bit times. Therefore, bit times 64-67 (the first bit time being 0) of each CRC window after the first will contain the CRC value for the previous window. There is no CRC transmission during the first 512 bit-time window after the link is initialized, and the value of the transmitted CRC bits is not included in the CRC calculation for the current window. Thus each CRC window after the first is 516 bit times in length - 512 of which are included in the calculation of the CRC which will be transmitted in the next window. There is no indication on the bus that CRC information is being transmitted; it is the responsibility of the parties on both ends of the link to count cycles from the beginning of the first valid packet after link synchronization to determine the boundaries of the CRC windows. During transmission of the CRC, the CTL bit will be driven to a value of 1 (Control).

The polynomial used to generate the CRC is:
$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$.

The CRC is calculated by computing the remainder resulting from the division of the data by the CRC polynomial. The register used to perform the calculation is seeded with all 1's at the beginning of each CRC window. Note that in the classical CRC definition, 32 0's are appended to the end of the data word before performing the division. This is not done in LDT. The CRC bits are inverted before being transmitted on the link in order to catch a wider range of bit errors.

All link errors, once detected, are fatal, and must be assumed to have corrupted both data and control information. If the CRC Flood Enable CSR bit is set, the detecting node will drop all operations in progress, and drive Sync packets continuously on all outgoing links. It will ignore all incoming packets from the fabric. It will also set the corresponding Link Failure CSR bit.

A CRC testing mode is also defined. Writing a 1 to the CRC Start Test bit of the Link Control register will cause the transmitter to enter CRC diagnostic mode. The transmitter will begin by issuing a Nop packet with its Diag bit set, which instructs the receiver to ignore the CAD and CTL lines for the following 512 bit times in each byte lane, not counting the bit times allotted to CRC stuffing.. The transmitter may then drive any pattern it wants on the CAD and CTL lines (other than during CRC stuffing), even to the extent of allowing CTL to change state between arbitrary bit times, with one exception. The test pattern may not

contain 4 consecutive bit times of all 1s on any byte lane (CAD and CTL lines), as that would be interpreted by the receiver as a sync packet. How the transmitter decides what to transmit as a test pattern is beyond the scope of this specification. CRC is still generated and checked for the interval, and CRC stuffing occurs normally, but the received data is ignored, and packet generating and tracking state machines are suspended in the state they were in when the diagnostic Nop was received. CRC errors detected during this time will be logged by setting the CRC Error bits, and will be treated as fatal if the CRC Flood Enable bit is set. If the CRC Force Error bit is set when the CRC Start Test bit is set, the test pattern will contain at least one CRC error in each active byte lane. When the test interval has completed, and the last CRC covering any part of the test interval has been stuffed, hardware will clear the CRC Start Test bit in the transmitter. Packet transmission will resume from the suspended state, which may be in the middle of a data packet.

11.1.2 Response Errors

Nodes generating TgtDone and RdResponse packets may set an error bit in these responses when an internal error condition occurs that may be localized to the transaction that the response packet is part of. The conditions that would cause this are beyond the scope of this specification. Response errors will also be generated when a non-posted request traverses the entire LDT chain without being accepted by any unit.

When an error bit is set in a response, the transaction that the response was a part of is considered to have failed. If the transaction was a read, the returning data may not be used. If the transaction was a write, the target location must be assumed to have gone to an undefined state.

In all cases, the goal is to complete the transaction protocol in the fabric, while returning the error notification to the requester. If the response is being returned to the requester, the latter is accomplished. The requester will issue any handshakes necessary to complete the transaction.

RdResponse packets with the error bit set still transmit the expected quantity of data, even though it may not be used by the destination.

11.2 Error Handling

Once detected, errors may cause one of several actions (in order of severity):

1. Signal requester via response error bit. It is then the responsibility of the requester to determine the appropriate action. This is only appropriate for transactions that can be localized to a single transaction, which can be completed and retired without hanging any fabric resources.
2. The error detector may generate an NMI to a CPU.
3. The error detector may broadcast Sync packets. This will bring the fabric down and require a hard reset to reinitialize the fabric.

BOZO: Clearly, this still needs work.

11.3 Sync Packets

Sync packets are used to broadcast fatal error information within all fabrics. Nodes detecting fatal errors drive their outgoing CAD and CTL bits on all links to logic 1, and keep them there. This will be recognized by devices at the other ends of the links as a sync packet, even if the nodes are out of sync or the clock has been corrupted. The node driving the error drops all transactions currently in flight, and ignores any packets received from its incoming links. CRC is not generated on links transmitting sync packets, nor is it checked on incoming links on which a sync packet has been detected.

Nodes receiving sync packets behave in exactly the same fashion as nodes detecting fatal errors, and propagate the error out all of their outgoing links. As sync packets propagate around the system, nodes will shut down, and activity in the fabric will cease. This state is stable, and will persist until a warm reset occurs.

12 Clocking

LDT systems consist of devices connected by LDT links. Devices within an LDT fabric may or may not be clocked by clocks derived from the same frequency source. Section 12.1 describes the requirements for LDT device clock sources. In addition to the clock source(s) for each device, there are two other out-of-band signals used by LDT devices: PwrOk and Reset_L. Section 13.2 describes how these signals are used.

12.1 Clock Requirements: Synchronous vs Asynchronous LDT Devices

Each LDT device has a dedicated frequency reference input pin that is used to generate the link output clock(s) for that device. The frequency of this reference may or may not be the same as the link clock frequency, and the means by which the link clock frequency is determined is device-specific and outside the scope of this specification.

Two classes of LDT devices are defined. A **synchronous (sync)** device requires the LDT device on the other side of the link to generate its clocks from the same frequency reference. An **asynchronous (async)** device does not require the device on the other side of the link to use the same reference – the reference inputs may be different frequencies, and may be sourced from different time bases. The only requirement is that the receivers at each end must have sufficient receive bandwidth to be able to sink the maximum bandwidth generated by the far transmitter. In order to cope with frequency error due to running nominally matched transmitter/receiver pairs from different frequency sources, the transmitter's raw bandwidth may exceed the receiver's core bandwidth by up to 1 part in 512. See section 12.3 for the suggested scheme for handling this situation.

The table below shows the supported combination of sync and async devices.

Device A	Device B	Same Reference?	Supported?
Sync	Sync	Y	Y
Sync	Sync	N	N
Sync	Async	Y	Y
Sync	Async	N	N
Async	Aync	Y	Y
Async	Async	Y	Y

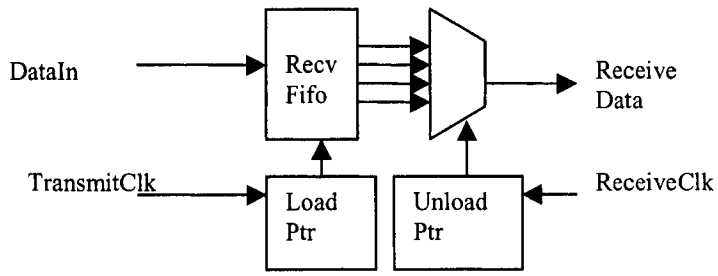
12.2 Receive FIFO initialization

Each LDT receiver contains a FIFO that is clocked by the transmitter. The FIFO is sized to absorb the *dynamic variation* between the transmitter's clock and the receiver's clock. The sources of this dynamic variation are:

- Temperature
- Voltage (either of the transmitter or the receiver)
- Accumulated phase error in a PLL
- Noise which affects the clock and data in the same way. (If the noise affected clock and data differently then this would affect the maximum bit rate, not the buffer depth.)

The following are **not** sources of dynamic variation but are traditionally considered to be contributors to clock skew:

- Process variation
- Etch length difference from a common clock source
- Anything else?



For links that are wider than 8 bits, the FIFO absorbs the difference in delay between 8-bit segments of the link. For links in which the two connected devices are clocked by different sources the FIFO absorbs the frequency variation. Also, the FIFO may be used to provide buffering between a narrow high-speed link and wider slower datapath inside a receiver.

Clocking successive bit-times into different FIFO entry flops serves to increase the valid time of each flop. Before operating the link, the unload pointer must be loaded with a value that provides adequate setup and hold times for sampling FIFO entries into the receiver's time domain, while at the same time minimizing the latency of the transfer. This is accomplished using the synch packets.

BOZO:

- Do we specify the minimum buffer depth that LDT devices must support? The current thinking is that this will be tied to the maximum link frequency that the device will support. Need to specify this.
- How does an instance of an LDT device know what the relationship between the load pointer and the unload pointer should be for that system? The current thinking is that this will also be tied to the frequency at which the link operates. This makes things simpler at the expense of increasing latency for some systems. Need to specify this.

BOZO: add details to this section which describe the algorithm and recommended implementation for initializing the receive FIFO.

12.3 Systems with Multiple Time Bases

In the case where the devices on each side of a link are clocked different sources the relationship between the FIFO load pointer and unload pointer will have to be adjusted. If the receive clock is faster than the transmit clock, then the unload pointer will occasionally have to be frozen. If the receive clock is slower than the transmit clock, we ensure that the transmitter does not overrun the receiver as follows.

The frequencies on either side of the link are specified to differ by no more than 1 part in 512. This is less than the rate at which CRC bits are inserted onto the link (4 parts in 512). CRC bits are sent by the transmitter and recomputed by the receiver. The receiver recomputes the CRC bits from the packet stream that is registered into the receiver's clock domain from the receive FIFO. The CRC bits are not placed directly into the receive FIFO, however. Instead the CRC bits are placed into dedicated flops which are clocked by the transmit clock, and evaluated by receive clock logic only after sufficient time as passed to ensure that these flops may be reliably sampled. Since the CRC bits only appear every 512 bit times there is a huge sample window for these flops. By not placing CRC bits into the receive FIFO, and hence not incrementing the unload pointer, we ensure that the receiver can always "catch up" to the transmitter.

BOZO: I decided that including specific circuits in the body of the spec is not appropriate. This stuff belongs in an app note somewhere.

13 Reset and Initialization

13.1 Definition of Reset

There are two types of reset defined at the fabric level:

1. Cold Reset – Node logic is reset. All links are reset. UnitIDs are assigned. All CSRs are reset. Cold reset is caused by the deassertion of PwrOK together with the assertion of Reset_L. Note that this sequence may be initiated under software control.
2. Warm Reset – Same as Cold Reset, except that CSRs defined to be "persistent" (expected to be mostly error state) are not reset. Warm reset is caused by asserting Reset_L and keeping PwrOk asserted. It may be initiated under software control.

The means by which PwrOk and Reset_L are generated within a specific system are outside of the scope of this specification.

13.2 System Powerup, Reset and Low-Level Link Initialization

For a cold reset sequence, PwrOk is asserted at least 1 ms after the power and clock sources for all LDT devices have become stable. The state of Reset_L is undefined before PwrOk is asserted and so it should be combined with PwrOK to generate internal resets. It must remain asserted for at least 1 ms beyond the assertion of PwrOk.

For a warm reset sequence, Reset_L must be asserted for at least 1 μs.

While Reset_L is asserted, each LDT device drives its outbound link(s) to the following state

Signal	State During Reset
CLK	Toggling
CTL	Logic 0
CAD[7:0]	Logic 1
CAD[31:8] (if present)	Logic 1

Note that this state does not correspond to any particular LDT packet type.

The link initialization sequence for two devices which share an LDT link involves transmitting and receiving values on the CTL and CAD signals.

A device-specific period of time after the deassertion of Reset_L each sync device asserts its CTL signal, initiating a sync sequence.. The assertion of the CTL signal serves to indicate to the device at the other side of the link that this device is ready to initialize the link. Devices perform whatever device-specific functions they may require between the time Reset_L is deasserted and the time they assert CTL. This may include ramping their internal clocks to full frequency, and reading configuration state from off-chip.

When a device has sampled the assertion of both its own CTL wire and the CTL wire driven from the other device on a rising CLK edge, it drives the Sync packet for at least 16 more bit times and then inverts both CAD and CTL.

The deassertion of the incoming CTL/CAD signals across a rising CLK edge is used in the transmit clock domain within each receiver to initialize the load pointer. The deassertion of the incoming CTL/CAD signals is synchronized to the core clock domain and used to initialize the unload pointer within each receiver. The length and uncertaintly of this synchronizer must be included to determine the proper relationship between the load pointer and the unload pointer. Note that CTL cannot be used to initialize the pointers for byte lanes other than 0 in a multi-byte link, as CTL only exists within byte 0's transmit clock domain.

Each device continues to drive this state on its outbound links for a minimum of 512 bit times, independent of the width of the link.

Each device then drives the CAD lines to logic 1 across a rising CLK edge, while leaving the CTL line deasserted, for exactly 4 bit times. The transition from all CAD lines deasserted to all CAD lines asserted serves to frame incoming packets. The first bit time after these 4 must have CTL asserted, and is both the first bit time of a new command packet, and the first bit time of the first CRC window. It will also occur across a rising CLK edge.

The entire sequence is shown in the table below:

CTL	CAD[7:0]	Notes
0	0xFF	• Value held during reset
1	0xFF	• CTL asserts device-specific time after Reset deasserts • Pattern held at least 16 bit times after both devices sample assertion of both CTL wires.
0	0x00	• 1->0 transition on incoming CTL/CAD initializes load pointer in transmit clock time domain • 1->0 transition on incoming CTL/CAD synchronized to core clock and used to initialize unload pointer in receive clock time domain • Pattern held for 512 bit times
0	0xFF	• 0->1 transition on CAD serves to define frame incoming packets • pattern held for 4 bit times
1	??	• 0->1 transition on CTL defines start of first control packet and represents first bit time of first CRC window

Using the initialization sequence as defined in the above section, sync and async devices can inter-operate as long as they share a common input clock.

13.3 I/O Fabric Initialization

I/O fabric initialization is largely software driven. A sample initialization sequence consists of the following steps:

1. One or more reset sequence initiators assert Reset_L (and possibly deassert PwrOk). Each initiator must sequence the PwrOk and Reset_L signals according to the rules defined in section 13.2. Multiple initiators may or may not release (deassert) Reset_L at the same time. In any event, the last initiator to release Reset_L determines when the initialization sequence begins. Note this means that each initiator must sample as well as drive Reset_L.

2. The low level link initialization sequence described in section 13.2 takes place between each node in the fabric.

3. Each node sends buffer-release packets to inform the transmitter(s) to which it connects how many buffers it contains.

In a double hosted chain, the host bridge at one end of the chain is designated the master host bridge, and the other the slave. How a host bridge determines whether it is a master or slave is outside the scope of this spec.

4. The slave host bridge, if any, sets a timer, and goes to sleep. The duration of the timer is implementation-specific. The master host bridge proceeds with the initialization sequence. At the beginning of the sequence, nextFreeID = 01h.

5. The master host bridge checks the Initialization Complete bit for its outgoing link, to determine if a device has been detected at the other end of the link. It also checks the CRC Error bits to see if the link has taken any errors since reset. If there is no device, or the link is taking errors, chain sizing is complete; proceed to step 8.

6. The bridge issues CSR accesses to unitID 00h, which is the Base Unit ID that all devices assume at reset, and is also responded to by host bridges. The reads are responded to by the first uninitialized device on the chain. Performing a write to the command register (without changing any fields) will cause the Master Host bit to get set, which indicates which link on that device pointing toward the host bridge. By polling the CRC error bits for that link, software can determine that that device is not seeing CRC errors on the link between it and the host. Software can then set the CRC Fatal Enable bit for the link from both ends. If the link is taking CRC errors, chain sizing is complete; proceed to step 8.

7. By reading the Class Code, Vendor ID, and Device ID, software can determine what type of device it is talking to. If it has reached an LDT device, it writes the Base Unit ID with nextFreeID, and increments nextFreeID by the Unit Count value of the device. Now that the device will no longer respond to accesses to 00h, the process can be repeated for the next link, starting back at step 5. If a slave host bridge has been reached, software sets the DoubleEnded bit on both that bridge and the master host bridge, and proceeds to step 9 for link partitioning.

8. At this point, an end to the chain has been found without reaching another host bridge. Software sets the EndOfChain and TransmitOff bits for the last link in the chain, and fabric initialization is complete. If there is a bridge at the other end, and the sizing algorithm hasn't reached it due to a break in the chain, it will wake up when its timer expires, find its DoubleEnded bit clear, and size the chain from the other end to the break, starting at step 5. The final result will be two single-ended chains, each with a master host bridge.

9. At this point, the entire chain has been sized, and found to have host bridges at both ends. When the slave host bridge's timer expires, it will find its DoubleEnded bit set, and know that no sizing is required on its part. All intermediate devices will have their Master Host bit pointing towards the master host bridge. It is, however, expected, that software will wish to partition the devices in the chain between the two host bridges for load balancing reasons. If so, software must select the location at which it wishes to break the chain, and access the nodes on either side of the break from the host bridge on that side. First, the EndOfChain bit for the link to be broken is set from each side. When both devices are ignoring the link, the TransmitOff bit for each side may be set.

The initialization process may be made more robust by providing a facility to time out the CSR accesses used for sizing, in the event that a device fails to respond. This possibility is beyond the scope of this specification.

13.4 Link Widths and Initialization

Each link controller contains two registers relating to the width of the link:

- A register that is hardwired to a particular value based on the capabilities of that device, and indicates the width of the outbound and inbound parts of the link.
- A register that indicates the width at which the inbound and outbound links should actually operate. This register is reset by cold reset so that both input and outbound links operate in 8-bit mode.

At cold reset all links power-up and synchronize as 8-bit links. Firmware (known as BIOS in the x86 world) interrogates all the links in the system, reprograms all the links to the desired width, and then takes the system through a warm reset to change the link widths.

14 Electrical Specification

CAD, CTL, and CLK are differential signals, at least in the current implementation.

BOZO: Anybody ever going to put anything else here?

A Ordering Rules Of Supported I/O Protocols

LDT is intended to support connections to I/O bridges supporting a variety of I/O protocols. At the present time, there are 2 I/O bus protocols identified that we wish to support: PCI, and AGP, each of which has different ordering requirements, as described below. It is the intent, however, to provide sufficiently flexible ordering support so that other protocols may be supported in the future.

A.1 PCI

The PCI ordering rules are taken from the PCI Local Bus Specification, rev 2.2 (6/8/98 draft), Appendix E. See that spec for more information.

Row Pass Column?	Posted Memory Write (PMW)	Delayed Read Request (DRR)	Delayed Write Request (DWR)	Delayed Read Completion (DRC)	Delayed Write Completion (DWC)
PMW	No	Yes	Yes	Yes	Yes
DRR	No	Yes/No	Yes/No	Yes/No	Yes/No
DWR	No	Yes/No	Yes/No	Yes/No	Yes/No
DRC	No	Yes	Yes	Yes/No	Yes/No
DWC	Yes/No	Yes	Yes	Yes/No	Yes/No

No – indicates the subsequently issued transaction is not allowed to complete before the previous transaction to preserve ordering in the system.

Yes – indicates the subsequently issued transaction must be allowed to complete before the previous transaction, or deadlock may occur. Reasons for all 8 **Yes** entries are given in the PCI spec. However, it indicates that the 4 **Yes**'s in the first row are only required for backward compatibility with earlier revs of the spec. For the PMW/DRR and PMW/DWR cases, this is an additional reason. They are required to be **Yes** because the fourth row requires DRCs to be able to pass DRRs and DWRs. In the case where a DRR or DWR occurs, followed by a PMW, followed by a DRC, the PMW must pass the DRR/DWR in order to allow the DRC to do so, since the DRC may not pass the PMW.

Yes/No – indicates the subsequently issued transaction may be allowed to complete before the previous transaction. There are no ordering requirements between the transactions.

In addition, PCI supports the concept of performing multiple system reads due to a single PCI bus read, in anticipation of that data being needed soon; this is called PCI prefetching. Within a prefetch, reads are required to be strongly ordered.

PCI is also capable of generating operations with discontiguous byte masks. If it does this for requests that cross aligned doubleword boundaries, they must be broken on doubleword boundaries into multiple transactions on LDT. In this case, the requests must be strongly ordered, in ascending order by address.

A.2 AGP

These ordering rules are taken from the Accelerated Graphics Port (AGP) Interface Specification, rev 2.0, section 3.4. See that spec for more information.

AGP essentially consists of 3 separate channels, each with its own distinct ordering rules. No ordering is maintained between the 3 channels; traffic is completely independent. First, AGP contains a modified PCI channel, which maintains PCI ordering. The other 2 channels are called the High Priority (HP) and Low Priority (LP) AGP channels.

The ordering rules presented here for reads are somewhat different than what appears in the AGP spec. It defines ordering between reads in terms of the order that data is returned to the requesting device. We are

concerned here with the order in which the reads are seen at the target; the I/O bridges can reorder returning read data if necessary. This leads to a slightly relaxed set of rules.

HP AGP Ordering Rules
1. Writes may not pass writes.

LP AGP Ordering Rules
1. Reads (including flushes) may not pass writes.
2. Writes may not pass writes.
3. Fences may not pass other transactions, nor be passed by other transactions.

AGP may also generate requests with discontiguous byte masks, with the same rules as PCI.

B Mapping of Protocol Ordering Rules Onto LDT

Ordering requirements for request packets are determined by their requester, and follow the request all the way to the destination. Ordering rules for responses are also determined by the original requester, and are taken from the request packet.

The sections below give the mapping from the traffic types of each of the supported protocols to LDT packet fields.

B.1 CPU

CPUs should generate non posted writes for PCI IO (64KB + 3 bytes) and for configuration space. It is implementation specific as to whether CPUs generate posted or nonposted writes for memory mapped IO.

In order to safely implement the producer-consumer model in all configurations, CPU requests should follow PCI ordering rules, with the PassPW bit always clear on both requests and responses.

B.2 PCI

PCI Transaction Type	LDT Packet Type
Posted Memory Write (PMW)	WrSized, Posted, PassPW = 0
Delayed Read Request (DRR)	RdSized, PassPW = 0, RespPassPW = 0
Delayed Write Request (DWR)	WrSized, Non-posted, PassPW = 0
Delayed Read Completion (DRC)	TgtDone, PassPW = 0 (from request packet)
Delayed Write Completion (DWC)	TgtDone, PassPW = 0

PCI Read Line and Read Multiple requests that cause prefetches across a 64-byte boundary use sequence tagging, with each PCI Read being assigned a different non-zero sequence ID. Non-posted PCI operations with discontiguous byte masks that get broken into multiple operations on LDT must also be tagged as part of a single sequence. All other requests use a sequence ID of 0.

B.3 AGP

The three channels of AGP are all completely independent as far as ordering is concerned, so (for optimal performance) an LDT to AGP I/O bridge should assign each of these I/O streams to a separate unitID.

The PCI channel of AGP uses the PCI mapping above.

The LP and HP channels never accept requests, so there is no need to specify the ordering of returning responses with respect to requests.

HP AGP Transaction Type	LDT Packet Type
HP Write	WrSized, Posted, PassPW = 1

HP AGP Transaction Type	LDT Packet Type
HP Read	RdSized, PassPW = 1, RespPassPW = 1

HP Writes are placed in the posted request channel, while Reads are placed in the non-posted request channel. Within each of these virtual channels, a single sequence ID is used to force the traffic to remain strongly ordered.

The PassPW and RespPassPW bits are set for reads because they are independent of the write traffic in the channel. The PassPW bit for writes doesn't matter in a pure HP AGP channel, because all the posted writes in the channel are strongly ordered due to the sequence ID anyway. But, if traffic from this channel were ever mixed with another I/O stream, having it set would minimize the interaction between the two.

There are two possible mappings of LP AGP traffic onto LDT. The first puts all traffic in the LDT nonposted channel:

LP AGP Transaction Type	LDT Packet Type
LP Write	WrSized, Non-Posted, PassPW = 1
LP Read	RdSized, PassPW = 1, RespPassPW = 1
LP Flush	*None*
LP Fence	*None*

All transfers in the low-priority channel are in the same virtual channel (non-posted requests), and are all assigned to the same non-zero sequence ID, which keeps them strongly ordered. While this is a stronger ordering rule than required by AGP, it is sufficient. Since all transactions are strongly ordered, there is no need to do anything with a fence command.

Even though LP Writes are not posted in this mapping onto LDT, they can still be posted from the AGP point of view. The transaction can complete on the AGP bus without waiting for TgtDone on LDT. However, the I/O bridge must remember that TgtDone is outstanding and not retire the buffer or srcTag until it is received. Since the writes are not posted, there is also no need to issue an explicit LDT flush packet. The I/O bridge can simply wait for TgtDone to be received for all outstanding writes, and then complete the flush operation on the AGP bus.

The values of the PassPW and RespPassPW bits don't matter in this mapping of a pure LP AGP channel, because there are no posted writes in this channel in either direction on LDT. However, if the traffic in this channel were ever to be combined with another I/O stream, setting them both would minimize the interactions with that stream.

The second mapping of LP AGP onto LDT puts LP writes in the posted channel:

LP AGP Transaction Type	LDT Packet Type
LP Write	WrSized, Posted, PassPW = 0
LP Read	RdSized, PassPW = 0, RespPassPW = 1
LP Flush	Flush, PassPW = 0
LP Fence	*None*

No use of nonzero sequence IDs is required. Ordering between LP writes is maintained by the fact that they are in the posted channel with their PassPW bits clear. LP reads are prevented from passing LP writes for the same reason. Flush operation use the LDT flush packet. Fences still don't result in LDT packets being sent, but they do require action in this mapping. Since no operation can pass a write, fences only need to be concerned with preventing other operations from passing reads. Therefore, they can be implemented by stalling all subsequent requests until responses have been received to all outstanding read requests.

As above, the value of RespPassPW on reads is not important in a pure LP AGP channel, but setting it may ease some interaction problems in a mixed channel.

C Southbridges and Compatibility Busses

This appendix provides some considerations for including compatibility busses (such as ISA) and southbridges in LDT-based systems.

C.1 ISA Deadlock case

A system that contains an ISA DMA controller has a particular deadlock scenario which needs to be addressed. Since ISA does not support retry, a downstream posted write could block a response to a nonposted request from the ISA bus, causing deadlock. Therefore components in the system must cooperate to ensure that there are no downstream posted writes between the host bridge and the ISA bus when nonposted DMA is executed.

LDT provides a mechanism to support this requirement.

An LDT-PCI bridge would have PREQ#/PGNT# pins to connect to the southbridge. When PREQ# is asserted, the LDT-PCI bridge generates a non-posted write to a predefined address range with address bit 3 set, and the PassPW bit clear. When this "PREQ-assert" packet is received, the host bridge sets an internal bit that causes it to stall any posted writes to the LDT chain containing the southbridge. It then issues a TgtDone response with the PassPW bit clear. When this response returns to the LDT-PCI bridge, it knows that all posted writes in flight in its direction have been flushed, and it may assert PGNT# to the southbridge. The southbridge then issues its requests, confident that they will complete without deadlock. When they have completed, it deasserts PREQ#, which causes the LDT-PCI bridge to issue a posted write to the same predefined address range, with address bit 3 clear, causing the host to clear its internal bit. Note that this mechanism does not support peer to peer requests from the ISA bus, which may still result in deadlock.

See section 10 for the predefined address range that supports this mechanism

C.2 ISA Write Post Flushing

Some ISA bridges require the ability to know when all posted writes they have issued are guaranteed globally visible. This requirement is typically handled by having a WSC# (Write Snoop Complete) pin on the southbridge, which is asserted whenever all previously posted writes are guaranteed to be visible to all processors.

No direct LDT support is required for an LDT-PCI bridge to implement the WSC# bit. The LDT-PCI bridge can follow each posted write from the ISA bridge with an LDT flush command. The response to the flush guarantees that the posted write is globally visible. The LDT-PCI bridge may then keep its WSC# pin asserted whenever it has no flushes outstanding.

C.3 Subtractive Decoding

This section provides some considerations for including a subtractive decoding device or bridge in an LDT-based system.

Since all LDT devices in a chain do not sit on the same bus, LDT devices cannot, in general, perform subtractive decode by waiting to see what requests are not responded to by other devices on the chain. Subtractive decoding devices and bridges are supported on LDT through the use of the Compat bit. Hosts connecting to LDT I/O chains are required to have positive decode ranges for all LDT devices/bridges except for the subtractive device/bridge. Requests that don't match any of the positive ranges are routed to the chain containing the subtractive device (the "compatibility chain") with the Compat bit set. The compat bit indicates to the subtractive device that it should claim the request, regardless of address.

The requirement that positive ranges be defined for all other LDT targets on the compatibility chain will require extra, implementation-specific range registers to be defined for the host bridge if the compatibility chain is PCI Config Bus Number 0, as there will then be no bridge header for this bus. Alternatively, bus 0

may be used to refer to a "virtual" bus internal to the host bridge, with a subtractive bridge header to the compatibility LDT chain.

Another option for supporting subtractive decode in LDT for small systems is to place the subtractive decode device on the end of the (single-hosted) LDT chain. In that case, the subtractive device can safely assume that all requests which reach it are destined for it.

If the subtractive target on the LDT chain is a bridge to PCI, there are several additional issues.

The most straightforward approach is to build an LDT to PCI bridge which performs subtractive decode and implements a standard PCI bridge header. See the PCI to PCI Bridge Architecture Specification for a description of subtractive decoding PCI to PCI bridges.

- The bridge performs subtractive decode using the compat bit, as described above, for transactions which originate on the primary bus.
- The bridge performs positive decode for transactions which originate on the secondary bus. It forwards to its primary bus any transaction that originates on the secondary bus and does not fall inside the address ranges programmed into the bridge header. This implies that peer-to-peer transactions targeted at a subtractive decoding device on the secondary bus and sourced on either the secondary or subordinate busses are not supported.
- The secondary bus segment is by definition not bus 0, since configuration software will encounter a bridge header and number the bus accordingly. This may have BIOS implications if a southbridge is present on the secondary bus.

Some systems may require that the southbridge be on bus 0, which is not allowed to be behind another bridge. Therefore, another approach may be used which allows the PCI bus that contains the subtractive decode device to be configured as bus 0. In this approach, the LDT to PCI bridge implements a function header rather than a bridge header.

- The bridge performs subtractive decode for transactions which originate on the primary bus, using the Compat bit.
- The bridge claims all transactions that originate on the secondary bus and forwards them to the primary bus, but it does not do this subtractively. This implies that peer to peer transactions that are targeted at devices on the secondary bus and sourced on either the secondary or subordinate busses are not supported.
- Any type 1configuration access targeted at bus 0 and forwarded down this LDT chain by the host would have its Compat bit set. Hence it will reach this bridge. The bridge will decode the request, recognize that it is targeted at bus 0, and forward a type 0 configuration access to the secondary bus.

It is worth noting that a system which sets up the subtractive decode path in hardware may talk over LDT to memory and I/O spaces owned by the subtractive device without requiring software initialization of the link; even though the devices on the LDT chain have not had their unitIDs programmed to nonzero values, the Compat bit will cause accesses to reach the subtractive device.

C.4 VGA Palette Snooping

The PCI Local Bus and Bridge Architecture Specifications define VGA Palette Snooping. This allows a device on the same bus as the device owning the VGA Palette range, or a bridge that forwards the VGA Palette range, to pick up write data as the access goes by.

No direct support for VGA Palette Snooping is provided in LDT. It may be supported at a level above the LDT protocol by designating an address range as an alias of the VGA Palette range, and having the host bridge generate a posted write to the alias as well as the write to the original address. The snooping device would have to recognize the aliased write, and translate it back to the VGA Palette range before forwarding or operating on it. The details of this mechanism are implementation-specific.

D Mapping Legacy 8259 (PIC) Interrupts In x86 Systems

The PIC is assumed to reside in the southbridge. When INTR out of the PIC is asserted, the southbridge generates an LDT interrupt request message as follows: MT=ExtInt; TM=edge; DM=physical; INTRDest=FFh; Vector=00h. The host bridge processes the interrupt request as a non-vectored interrupt as described in section **Error! Reference source not found.**. The CPU which services the interrupt request issues an interrupt acknowledge (RdSized) cycle to the southbridge. The interrupt vector is returned in the 8 LSBs of the RdResponse. The 24 MSBs are zero.

In order for interrupt ordering to be maintained relative to posted writes, it is necessary to make sure that all posted writes from a device have reached their destination before interrupts from that device are received. This requires that the PIC (and therefore the southbridge) be placed at the end of the LDT I/O chain, farthest away from the host. The southbridge must issue a fence packet ahead of the interrupt packet. This will guarantee that posted writes from all devices on the chain are flushed to the host bridge ahead of the interrupt.

Note that this also implies that PIC interrupts may not be used in systems with multiple LDT I/O chains, or double-hosted chains.